

(19)



(11)

EP 3 682 378 B1

(12)

EUROPEAN PATENT SPECIFICATION

(45) Date of publication and mention of the grant of the patent:
18.09.2024 Bulletin 2024/38

(21) Application number: **18778633.0**

(22) Date of filing: **30.08.2018**

(51) International Patent Classification (IPC):
G06N 3/084 (2023.01) **G06N 3/063** (2023.01)

(52) Cooperative Patent Classification (CPC):
G06N 3/084; G06N 3/063

(86) International application number:
PCT/US2018/048784

(87) International publication number:
WO 2019/055224 (21.03.2019 Gazette 2019/12)

(54) **SYSTEM AND METHOD FOR IMPLEMENTING NEURAL NETWORKS IN INTEGRATED CIRCUITS**

SYSTEM UND VERFAHREN ZUR IMPLEMENTIERUNG NEURONALER NETZWERKE IN INTEGRIERTEN SCHALTUNGEN

SYSTÈME ET PROCÉDÉ DE MISE EN OEUVRE DE RÉSEAUX NEURONAUX DANS DES CIRCUITS INTÉGRÉS

(84) Designated Contracting States:
AL AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HR HU IE IS IT LI LT LU LV MC MK MT NL NO PL PT RO RS SE SI SK SM TR

(30) Priority: **14.09.2017 US 201715705033**

(43) Date of publication of application:
22.07.2020 Bulletin 2020/30

(73) Proprietor: **Xilinx, Inc.**
San Jose, California 95124 (US)

(72) Inventors:
• **FRASER, Nicholas**
San Jose, California 95124 (US)
• **BLOTT, Michaela**
San Jose, California 95124 (US)

(74) Representative: **Zimmermann & Partner**
Patentanwälte mbB
Postfach 330 920
80069 München (DE)

(56) References cited:
WO-A1-2017/201506 WO-A1-2017/201506
CN-A- 106 228 240 CN-A- 106 228 240
CN-A- 107 153 873 CN-A- 107 153 873

- **MAX JADERBERG ET AL: "Decoupled Neural Interfaces using Synthetic Gradients", ARXIV.ORG, CORNELL UNIVERSITY LIBRARY, 201 OLIN LIBRARY CORNELL UNIVERSITY ITHACA, NY 14853, 18 August 2016 (2016-08-18), XP080720948**
- **WOJCIECH MARIAN CZARNECKI ET AL: "Understanding Synthetic Gradients and Decoupled Neural Interfaces", ARXIV.ORG, CORNELL UNIVERSITY LIBRARY, 201 OLIN LIBRARY CORNELL UNIVERSITY ITHACA, NY 14853, 1 March 2017 (2017-03-01), XP080753587**
- **PRIYA GOYAL ET AL: "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour", ARXIV.ORG, CORNELL UNIVERSITY LIBRARY, 201 OLIN LIBRARY CORNELL UNIVERSITY ITHACA, NY 14853, 8 June 2017 (2017-06-08), XP080768532**
- **YAMAN UMUROGLU ET AL: "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference", ARXIV.ORG, CORNELL UNIVERSITY LIBRARY, 201 OLIN LIBRARY CORNELL UNIVERSITY ITHACA, NY 14853, 1 December 2016 (2016-12-01), XP080742060, DOI: 10.1145/3020078.3021744**
- **HUIPING ZHUANG ET AL: "Fully Decoupled Neural Network Learning Using Delayed Gradients", ARXIV.ORG, CORNELL UNIVERSITY LIBRARY, 201 OLIN LIBRARY CORNELL UNIVERSITY ITHACA, NY 14853, 21 June 2019 (2019-06-21), XP081378928**

Note: Within nine months of the publication of the mention of the grant of the European patent in the European Patent Bulletin, any person may give notice to the European Patent Office of opposition to that patent, in accordance with the Implementing Regulations. Notice of opposition shall not be deemed to have been filed until the opposition fee has been paid. (Art. 99(1) European Patent Convention).

EP 3 682 378 B1

- ZHOUYUAN HUO ET AL: "Training Neural Networks Using Features Replay", ARXIV.ORG, CORNELL UNIVERSITY LIBRARY, 201 OLIN LIBRARY CORNELL UNIVERSITY ITHACA, NY 14853, 12 July 2018 (2018-07-12), XP081432571
- ZHOUYUAN HUO ET AL: "Decoupled Parallel Backpropagation with Convergence Guarantee", ARXIV.ORG, CORNELL UNIVERSITY LIBRARY, 201 OLIN LIBRARY CORNELL UNIVERSITY ITHACA, NY 14853, 27 April 2018 (2018-04-27), XP081240942
- SIRAJ RAVAL: "Synthetic Gradients Explained", YOUTUBE, 24 October 2017 (2017-10-24), pages 1 pp., XP054979563, Retrieved from the Internet <URL:https://www.youtube.com/watch?v=qirjknNY1zo> [retrieved on 20190729]
- EUGENE BELILOVSKY ET AL: "Decoupled Greedy Learning of CNNs", ARXIV.ORG, CORNELL UNIVERSITY LIBRARY, 201 OLIN LIBRARY CORNELL UNIVERSITY ITHACA, NY 14853, 24 January 2019 (2019-01-24), XP081364990
- MAX JADERBERG ET AL: "Decoupled Neural Interfaces using Synthetic Gradients", ARXIV.ORG, CORNELL UNIVERSITY LIBRARY, 201 OLIN LIBRARY CORNELL UNIVERSITY ITHACA, NY 14853, 18 August 2016 (2016-08-18), XP080720948
- WOJCIECH MARIAN CZARNECKI ET AL: "Understanding Synthetic Gradients and Decoupled Neural Interfaces", ARXIV.ORG, CORNELL UNIVERSITY LIBRARY, 201 OLIN LIBRARY CORNELL UNIVERSITY ITHACA, NY 14853, 1 March 2017 (2017-03-01), XP080753587
- PRIYA GOYAL ET AL: "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour", ARXIV.ORG, CORNELL UNIVERSITY LIBRARY, 201 OLIN LIBRARY CORNELL UNIVERSITY ITHACA, NY 14853, 8 June 2017 (2017-06-08), XP080768532
- YAMAN UMUROGLU ET AL: "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference", ARXIV.ORG, CORNELL UNIVERSITY LIBRARY, 201 OLIN LIBRARY CORNELL UNIVERSITY ITHACA, NY 14853, 1 December 2016 (2016-12-01), XP080742060, DOI: 10.1145/3020078.3021744

Description

TECHNICAL FIELD

[0001] Examples of the present disclosure generally relate to integrated circuits ("ICs") and, in particular, to neural networks in ICs.

BACKGROUND

[0002] Machine learning capabilities are fast evolving. For example, neural networks (NNs) may be used in both data centers and embedded applications. Typically, neural networks need to be trained on a sufficiently large dataset, and the training is performed on the basis of floating point arithmetic using general purpose graphics processing units (GPGPUs). Such training, depending on the size of a network and the size of the training datasets, may take a very long time. In general, a training dataset consists of input/output pairs. The goal of neural network training is to create a model which maps each input of the training set to the corresponding output. In real use cases, this model may also generalize to data which is not part of the training set.

[0003] In the publication "MAX JADERBERG ET AL: "Decoupled Neural Interfaces using Synthetic Gradients", ARXIV.ORG, CORNELL UNIVERSITY LIBRARY, 201 OLIN LIBRARY CORNELL UNIVERSITY ITHACA, NY 14853, 18 August 2016 (2016-08-18), XP080720948" decoupled neural interfaces using synthetic gradients are described, wherein sub-graphs are decoupled by using a modelled synthetic gradient in place of true back-propagated error gradients. In this way, the decoupled sub-graphs can be updated independently from each other and asynchronously. Further, feed-forward models are presented, where every layer is trained asynchronously.

[0004] Accordingly, it would be desirable and useful to provide an improved way of implementing neural network trainings.

SUMMARY

[0005] The invention is defined by the independent claims.

[0006] Preferred embodiments are defined by the dependent claims.

[0007] Other aspects and features will be evident from reading the following detailed description and accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

[0008]

FIG. 1 is a block diagram illustrating an exemplary architecture for an IC.

FIG. 2 is a block diagram illustrating an exemplary neural network system.

FIG. 3 is a block diagram illustrating an exemplary layer of a neural network system.

FIG. 4 is a block diagram illustrating a forward path processing element.

FIG. 5 is a block diagram illustrating a backward path processing element.

FIG. 6 is a block diagram illustrating a neural network system implementing a delayed model adaptation scheme.

FIG. 7 is a block diagram illustrating a layer of the neural network system of FIG. 6.

FIG. 8 is a flow diagram of a method for training a neural network system.

FIG. 9A is a block diagram illustrating a synthetic gradient generator.

FIG. 9B is a block diagram illustrating a synthetic gradient generator.

FIG. 10 is a block diagram illustrating a portion of a binarized neural network system implementing backpropagation calculation.

FIG. 11 is a block diagram illustrating a neural network system implemented using a plurality of computing units.

FIG. 12 is a block diagram illustrating a neural network system implemented using a single processing array.

FIG. 13 is a block diagram illustrating a neural network system implemented using two processing arrays.

45 DETAILED DESCRIPTION

[0009] Various embodiments are described hereinafter with reference to the figures. Like reference numerals refer to like elements throughout. Like elements will, thus, not be described in detail with respect to the description of each figure. It should also be noted that the figures are only intended to facilitate the description of the embodiments. In addition, an illustrated embodiment needs not have all the aspects or advantages shown. An aspect or an advantage described in conjunction with a particular embodiment is not necessarily limited to that embodiment and can be practiced in any other embodiments even if not so illustrated, or if not so explicitly described.

The advantages may be achieved independently in various embodiments or may be combined in yet other embodiments.

[0010] Before describing exemplary embodiments illustratively depicted in the several figures, a general introduction is provided to further understanding. An artificial neural network (also referred to as neural networks below) is a model of computation inspired by the structure of brain neural networks. The neural network includes an input layer, a plurality of hidden layers, and an output layer. Each layer may include one or more basic processing elements (PEs), also referred to as neurons. These layers and their neurons are connected to each other, through which the neural network may carry out highly complex computations. An activation (e.g., an output) of a particular neuron may be obtained by applying weights to the outputs of other neurons connected to that particular neuron, generating a weighted sum of the outputs of those other neurons, and apply an activation function to the weighted sum. In some embodiments, a backpropagation learning method may be used to calculate the error contribution of each neuron after a batch of data (e.g., in image recognition, multiple images) is processed. Such error contribution may be calculated as the gradients of a loss function, where the error is calculated at the output layer and distributed back through the network layers. The weights may then be updated using the gradients.

[0011] As discussed above, a neural network typically needs to be trained on a sufficiently large training dataset. The training dataset may include a plurality of subsets (batches). A typical training algorithm may have various dependencies. Such dependencies may result in stalling and cause difficulties for implementing the neural network using multiple accelerators. Due to the dependencies present between the forward path and the backward path of the neural network, in some embodiments, the forward path for each batch is executed in a layer by layer fashion to reduce the time difference between processing adjacent batches. Furthermore, the training may be performed based on floating point arithmetic using GPGPUs. However, such training may take a long time (e.g., hours to weeks) depending on the size of the neural network and the size of the training dataset.

[0012] It has been discovered that various techniques may be used to remove the dependencies in a training algorithm, which may prevent or reduce stalling and allow implementations using multiple accelerators (e.g., multiple GPUs, multiple FPGAs, multiple application-specific integrated circuits (ASIC)s, multiple accelerators (e.g., systolic array) within a single FPGA). For example, a delayed model adaptation scheme may be used to remove dependencies between activations (e.g., computed by the forward path) and gradients (e.g., computed by the backward path) during the training. Such a delayed model adaptation allows the forward path and the backward path of neural network training to be implemented in parallel. This is achieved by introducing a delay between the

output activations and the weight update and gradient calculations in each layer. This allows the calculations for each layer to be performed independently without different batches of inputs. For further example, synthetic gradients (e.g., gradient estimates generated by machine learning algorithms) may be used to provide gradients for each layer of the neural network, thereby eliminating the dependencies involving gradients during the training.

[0013] It has also been discovered that the training efficiency may be improved with little or no impact on the accuracy by removing the significant redundancy in neural networks based on floating point arithmetic. For example, data types with reduced precision (e.g., based on binary arithmetic) may be used in the training of the neural network. A neural network using binary data types (e.g., binary weights, binary activations, and/or binary gradients) may be referred to as a binarized neural network. Such a binarized neural network may be mapped efficiently to a programmable IC, such as a field programmable gate array (FPGA). Furthermore, custom data representations (e.g., representations using logarithmic arithmetic or fixed point arithmetic with or without stochastic quantization) may be used to improve the hardware implementations of the neural network.

[0014] With the above general understanding borne in mind, various embodiments for implementing a neural network are described below. Various advantages may be present in various applications of the present disclosure. No particular advantage is required for all embodiments, and different embodiments may offer different advantages. One of the advantages is that by removing the dependencies between layers of a neural network (e.g., by using a delayed model adaptation scheme and/or synthetic gradients), the neural network may be implemented using multiple compute units (e.g., accelerators), which may reduce the training time of the neural network. For example, the accelerators may be configured based on different computation and memory requirements of the corresponding layers, which may improve accelerator utilization.

[0015] Another advantage in some embodiments is that reduced precisions (e.g., binarized neural network) and custom data presentations (e.g., logarithmic arithmetic presentations) may be used in training the neural network, which may improve the training efficiency. Such reduced precisions and custom data presentations may be used various implementations of a neural network, e.g., implementations using a single accelerator (e.g., a single GPU, an FPGA including a single systolic array) or multiple accelerators (e.g., multiple GPUs, multiple FPGAs, a single FPGA including multiple systolic arrays).

[0016] Yet another advantage in some embodiments is that neural network training using FPGAs may use reduced power consumption (e.g., compared to GPUs), which may enable online training. Such online training may allow embedded devices of the FPGAs to continually adjust to environmental changes, and achieve accuracy

and throughput tradeoffs without re-synthesis of the hardware implementation of the neural network. Yet another advantage in some embodiments is that by using a pre-processing unit (e.g., on-chip or off-chip with hardware implementation of the neural network) to artificially augment incoming training datasets (e.g., by distorting, shading, rotating, scaling), higher training accuracy may be achieved.

[0017] Yet another advantage in some embodiments is that within each layer of the neural network, random number generators may be used to support random rounding and dropout. By using the various techniques described in detail below, more efficient neural network training is achieved, thereby increasing the productivity of data scientists and improving the accuracy of neural networks.

[0018] Yet another advantage in some embodiments is that by removing the data dependency as imposed by the requirement of completing the computation of the full forward path before computing the full backward path, a higher degree of parallelism is introduced that can be exploited in accelerator architectures (e.g. dataflow).

[0019] Yet another advantage is that by introducing a delay in the weights in the forward paths and the updates calculated using the gradients, training can work successfully by using estimates of the gradients.

[0020] By removing the data dependency, a higher degree of parallelism is introduced that can be exploited in accelerator architectures (e.g. dataflow).

[0021] Because one or more of the above-described embodiments are exemplified using a particular type of IC, a detailed description of such an IC is provided below. However, it should be understood that other types of ICs may benefit from one or more of the embodiments described herein.

[0022] Programmable logic devices ("PLDs") are a well-known type of integrated circuit that can be programmed to perform specified logic functions. One type of PLD, the field programmable gate array ("FPGA"), typically includes an array of programmable tiles. These programmable tiles can include, for example, input/output blocks ("IOBs"), configurable logic blocks ("CLBs"), dedicated random access memory blocks ("BRAMs"), multipliers, digital signal processing blocks ("DSPs"), processors, clock managers, delay lock loops ("DLLs"), and so forth. As used herein, "include" and "including" mean including without limitation.

[0023] Each programmable tile typically includes both programmable interconnect and programmable logic. The programmable interconnect typically includes a large number of interconnect lines of varying lengths interconnected by programmable interconnect points ("PIPs"). The programmable logic implements the logic of a user design using programmable elements that can include, for example, function generators, registers, arithmetic logic, and so forth.

[0024] The programmable interconnect and programmable logic are typically programmed by loading a

stream of configuration data into internal configuration memory cells that define how the programmable elements are configured. The configuration data can be read from memory (e.g., from an external PROM) or written into the FPGA by an external device. The collective states of the individual memory cells then determine the function of the FPGA.

[0025] Another type of PLD is the Complex Programmable Logic Device, or complex programmable logic devices (CPLDs). A CPLD includes two or more "function blocks" connected together and to input/output ("I/O") resources by an interconnect switch matrix. Each function block of the CPLD includes a two-level AND/OR structure similar to those used in Programmable Logic Arrays ("PLAs") and Programmable Array Logic ("PAL") devices. In CPLDs, configuration data is typically stored on-chip in non-volatile memory. In some CPLDs, configuration data is stored on-chip in non-volatile memory, then downloaded to volatile memory as part of an initial configuration (programming) sequence.

[0026] In general, each of these programmable logic devices ("PLDs"), the functionality of the device is controlled by configuration data provided to the device for that purpose. The configuration data can be stored in volatile memory (e.g., static memory cells, as common in FPGAs and some CPLDs), in non-volatile memory (e.g., FLASH memory, as in some CPLDs), or in any other type of memory cell.

[0027] Other PLDs are programmed by applying a processing layer, such as a metal layer, that programmably interconnects the various elements on the device. These PLDs are known as mask programmable devices. PLDs can also be implemented in other ways, e.g., using fuse or antifuse technology. The terms "PLD" and "programmable logic device" include but are not limited to these exemplary devices, as well as encompassing devices that are only partially programmable. For example, one type of PLD includes a combination of hard-coded transistor logic and a programmable switch fabric that programmably interconnects the hard-coded transistor logic.

[0028] As noted above, advanced FPGAs can include several different types of programmable logic blocks in the array. For example, FIG. 1 illustrates an exemplary FPGA architecture 100. The FPGA architecture 100 includes a large number of different programmable tiles, including multi-gigabit transceivers ("MGTs") 101, configurable logic blocks ("CLBs") 102, random access memory blocks ("BRAMs") 103, input/output blocks ("IOBs") 104, configuration and clocking logic ("CONFIG/CLOCKS") 105, digital signal processing blocks ("DSPs") 106, specialized input/output blocks ("I/O") 107 (e.g., configuration ports and clock ports), and other programmable logic 108 such as digital clock managers, analog-to-digital converters, system monitoring logic, and so forth. Some FPGAs also include dedicated processor blocks ("PROC") 110.

[0029] In some FPGAs, each programmable tile can

include at least one programmable interconnect element ("INT") 111 having connections to input and output terminals 120 of a programmable logic element within the same tile, as shown by examples included at the top of FIG. 1. Each programmable interconnect element 111 can also include connections to interconnect segments 122 of adjacent programmable interconnect element(s) in the same tile or other tile(s). Each programmable interconnect element 111 can also include connections to interconnect segments 124 of general routing resources between logic blocks (not shown). The general routing resources can include routing channels between logic blocks (not shown) comprising tracks of interconnect segments (e.g., interconnect segments 124) and switch blocks (not shown) for connecting interconnect segments. The interconnect segments of the general routing resources (e.g., interconnect segments 124) can span one or more logic blocks. The programmable interconnect elements 111 taken together with the general routing resources implement a programmable interconnect structure ("programmable interconnect") for the illustrated FPGA.

[0030] In an example implementation, a CLB 102 can include a configurable logic element ("CLE") 112 that can be programmed to implement user logic plus a single programmable interconnect element ("INT") 111. A BRAM 103 can include a BRAM logic element ("BRL") 113 in addition to one or more programmable interconnect elements. Typically, the number of interconnect elements included in a tile depends on the height of the tile. In the pictured example, a BRAM tile has the same height as five CLBs, but other numbers (e.g., four) can also be used. A DSP tile 106 can include a DSP logic element ("DSPL") 114 in addition to an appropriate number of programmable interconnect elements. An IOB 104 can include, for example, two instances of an input/output logic element ("IOL") 115 in addition to one instance of the programmable interconnect element 111. As will be clear to those of skill in the art, the actual I/O pads connected, for example, to the I/O logic element 115 typically are not confined to the area of the input/output logic element 115.

[0031] In the example of FIG. 1, an area (depicted horizontally) near the center of the die (e.g., formed of regions 105, 107, and 108 shown in FIG. 1) can be used for configuration, clock, and other control logic. Column 109 (depicted vertically) extending from this horizontal area or other columns may be used to distribute the clocks and configuration signals across the breadth of the FPGA.

[0032] Some FPGAs utilizing the architecture illustrated in FIG. 1 include additional logic blocks that disrupt the regular columnar structure making up a large part of the FPGA. The additional logic blocks can be programmable blocks and/or dedicated logic. For example, PROC 110 spans several columns of CLBs and BRAMs. PROC 110 can include various components ranging from a single microprocessor to a complete programmable

processing system of microprocessor(s), memory controllers, peripherals, and the like.

[0033] In one aspect, PROC 110 is implemented as a dedicated circuitry, e.g., as a hard-wired processor, that is fabricated as part of the die that implements the programmable circuitry of the IC. PROC 110 can represent any of a variety of different processor types and/or systems ranging in complexity from an individual processor, e.g., a single core capable of executing program code, to an entire processor system having one or more cores, modules, co-processors, interfaces, or the like.

[0034] In another aspect, PROC 110 is omitted from architecture 100, and may be replaced with one or more of the other varieties of the programmable blocks described. Further, such blocks can be utilized to form a "soft processor" in that the various blocks of programmable circuitry can be used to form a processor that can execute program code, as is the case with PROC 110.

[0035] The phrase "programmable circuitry" can refer to programmable circuit elements within an IC, e.g., the various programmable or configurable circuit blocks or tiles described herein, as well as the interconnect circuitry that selectively couples the various circuit blocks, tiles, and/or elements according to configuration data that is loaded into the IC. For example, portions shown in FIG. 1 that are external to PROC 110 such as CLBs 102 and BRAMs 103 can be considered programmable circuitry of the IC.

[0036] In some embodiments, the functionality and connectivity of programmable circuitry are not established until configuration data is loaded into the IC. A set of configuration data can be used to program programmable circuitry of an IC such as an FPGA. The configuration data is, in some cases, referred to as a "configuration bitstream." In general, programmable circuitry is not operational or functional without first loading a configuration bitstream into the IC. The configuration bitstream effectively implements or instantiates a particular circuit design within the programmable circuitry. The circuit design specifies, for example, functional aspects of the programmable circuit blocks and physical connectivity among the various programmable circuit blocks.

[0037] In some embodiments, circuitry that is "hard-wired" or "hardened," i.e., not programmable, is manufactured as part of the IC. Unlike programmable circuitry, hardwired circuitry or circuit blocks are not implemented after the manufacture of the IC through the loading of a configuration bitstream. Hardwired circuitry is generally considered to have dedicated circuit blocks and interconnects, for example, that are functional without first loading a configuration bitstream into the IC, e.g., PROC 110.

[0038] In some instances, hardwired circuitry can have one or more operational modes that can be set or selected according to register settings or values stored in one or more memory elements within the IC. The operational modes can be set, for example, through the loading of a configuration bitstream into the IC. Despite this ability, hardwired circuitry is not considered programmable cir-

cuitry as the hardwired circuitry is operable and has a particular function when manufactured as part of the IC.

[0039] FIG. 1 is intended to illustrate an exemplary architecture that can be used to implement an IC that includes programmable circuitry, e.g., a programmable fabric. For example, the numbers of logic blocks in a row, the relative width of the rows, the number and order of rows, the types of logic blocks included in the rows, the relative sizes of the logic blocks, and the interconnect/logic implementations included at the top of FIG. 1 are purely exemplary. For example, in an actual IC, more than one adjacent row of CLBs is typically included wherever the CLBs appear, to facilitate the efficient implementation of user logic, but the number of adjacent CLB rows varies with the overall size of the IC. Moreover, the FPGA of FIG. 1 illustrates one example of a programmable IC that can employ examples of the interconnect circuits described herein. The interconnect circuits described herein can be used in other types of programmable ICs, such as CPLDs or any type of programmable IC having a programmable interconnect structure for selectively coupling logic elements.

[0040] It is noted that the IC that may implement the neural network is not limited to the exemplary IC depicted in FIG. 1, and that IC having other configurations, or other types of IC, may also implement the neural network system.

[0041] Referring to FIG. 2, illustrated is an exemplary neural network system 200. The neural network system 200 includes a preprocessing unit 202, N layers including layers 204-0 through 204-(N-1), and a loss computation unit 210. The layers may be implemented using programmable logic devices, DSP blocks, etc. In an example, the neural network system 200 including the N layers may be implemented in a single IC. In another example, multiple ICs may be used to implement the neural network system 200, where each IC may implement one or more layers of the neural network system 200. Each layer may include one or more neurons, each neuron having its corresponding forward path processing element (PE) and backward path PE, which will be described in detail below. The first layer 204-0 may receive the input training set, and may be referred to as the input layer 204-0. The input training set may include a sequence of batches having batch indexes 0, 1, 2, 3, ..., B-1, where B is the number of batches in the input training set. A batch having an index b may be referred to as the bth batch. The last layer 204-(N-1) may provide an output for a corresponding input received by the input layer 204-0, and may be referred to as the output layer 204-(N-1). Layers 204-1 through 204-(N-2) may be referred to as hidden layers.

[0042] A preprocessing unit 202 may receive an input training set 220, artificially augment batches in the input training set 220 (e.g., by performing distorting, shading, rotating, scaling, cropping, and other applicable processes), and generates the preprocessed training set 222 including augmented batches. In some examples, the augmented batch has a size that is greater or equal to that

of the corresponding original batch. Such preprocessing may be performed when the data rate of input training set 220 (e.g., images) is unable to fully utilize the computational capabilities of target device(s) implementing the neural network system 200. The data rate of the input training set 220 may be affected by various processes for capturing and providing the input training set 220. For example, it may be affected by the speed of a source capture device (e.g., a camera, a microphone) for capturing the input training set 220 (e.g., images). For further example, it may be affected by a bottleneck caused by an external memory used to provide input training set 220. The preprocessing may be performed when such a data rate of the input training set 220 is lower than a threshold data rate determined based on the computational capabilities of the target device(s). In some examples, after the preprocessing, the preprocessed training set 222 has a larger size than the input training set 220. Such a preprocessed training set 222 with an increased size may improve the quality of the trained neural network (e.g., by improving the generality of models created by the neural network training), and better utilize the computational capabilities of the target device(s) implementing the neural network system.

[0043] The neural network system 200 may implement various training algorithms. In some embodiments, the neural network system may use a backpropagation algorithm. Consider a neural network has weights $W(b)$ for the bth batch, and provides an estimation function $f_b(x_i)$ based on the weights $W(b)$, where x_i is an input to be estimated (i.e., an image to be recognized). For such a neural network, the backpropagation algorithm includes two processes, a forward path (forward propagation) process, and a backward path (backward propagation) process.

[0044] A forward path process includes computing activations 206-0 through 206-(N-1) for all layers 204-0 through 204-(N-1). For a preprocessed training set 222 including M samples, the input/output pairs may be denoted as $\{x_i, y_i\}$, where $i=0, \dots, M-1$. An estimate of an output $\tilde{y}_i$ may be calculated as:

$$\tilde{y}_i = f_b(x_i),$$

where f_b refers to a model described by a given neural network for weights calculated for a given bth batch. The process for calculating the output may include first calculating the activations 206-0 by layer 204-0, passing the activations 206-0 from layer 204-0 to the next layer 204-1, calculating the activations 206-1 by layer 204-1, and so on until the last layer 204-(N-1) calculates activations 206-(N-1). The activations 206-(N-1) may then be sent to a loss computation unit 210.

[0045] The backward path process includes computing gradients 208-N through 208-1. In the example of FIG. 2, after receiving the activations 206-(N-1) from the last layer 204-(N-1), the loss computation unit 210 may use

a predetermined loss function to compute gradients 208-N based on the activations 206-(N-1) and the preprocessed training set 222. In an example, the predetermined loss function may be a sub-differential loss function, for example, a squared hinge loss function provided as:

$$l(y) = \max(0, 1 - \tilde{y}_i y_i)^2.$$

[0046] The loss computation unit 210 may then send the gradients 208-N to the last layer 204-(N-1). A layer 204-i may use the loss function to calculate its gradients 208-i based on gradients 208-(i+1) received from the next layer 204-(i+1) and activations 206-(i-1) received from the preceding layer 204-(i-1). The calculated gradients 208-i may then be passed to the preceding layer 204-(i-1), which may be used to calculate the gradient at each weight by iteratively applying the chain rule, which is a formula for computing the derivative of the composition of two or more functions. Such computation of gradients continues until the gradients of weights in all layers 204-(N-1) to 204-0 have been calculated.

[0047] In some embodiments, during the backward path process, gradients of weights may be used to update the weights using a gradient descent algorithm. In an example, in a particular layer, the weight update process may be provided as follows:

$$W(b+1) = W(b) - \mu \text{Grad}(b),$$

where $W(b)$ is the weights for the b^{th} batch, $\text{Grad}(b)$ is the gradients for the b^{th} batch, μ is a scaling factor referred to as a learning rate or a step size, and $W(b+1)$ are the weights for the $(b+1)^{\text{th}}$ batch.

[0048] Referring to FIGS. 3, 4, and 5, an exemplary layer 204-i of the neural network system 200 is illustrated. FIG. 3 illustrates a layer 204-i including a forward path unit 300 for performing the forward path process of the backpropagation algorithm at the layer 204-i, and a backward path unit 302 for performing the backward path process of the backpropagation algorithm at the layer 204-i. FIG. 4 illustrates an exemplary forward path processing element (PE) of the forward path unit 300. FIG. 5 illustrates an exemplary backward path PE of the backward path unit 302.

[0049] Referring to the example of FIG. 3, a layer 204-i includes a forward path unit 300 and a backward path unit 302. The forward path unit 300 includes a buffer 304 and P forward path processing elements (PEs) 306-0 through 306-(P-1), where P is an integer equal to or greater than one. The buffer 304 receives activations 206-(i-1) from the preceding layer 204-(i-1), and sends the activations 206-(i-1) to P forward path PEs 306-0 through 306-(P-1). The forward path PEs 306-0 through 306-(P-1) may perform an activation generation function to generate activations 206-i based on the activations 206-(i-1) from the preceding layer 204-(i-1), and send the acti-

vations 206-i to the next layer 204-(i+1) (e.g., through a buffer 308). In some embodiments, each forward path PE corresponds to a neuron of the layer 204-i. As such, the number P of forward path PEs in the forward path unit 300 may control the number of neurons of the layer 204-i that may be computed in parallel in the forward path process. In some embodiments, the forward path PEs 306-0 through 306-(P-1) may receive weights 318 from corresponding backward path PEs 310-0 through 310-(P-1) respectively, and perform the activation generation function based on the updated weights 318 and the activations 206-(i-1).

[0050] In the example of FIG. 3, the backward path unit 302 includes a buffer 312 and P backward path PEs 310-0 through 310-(P-1). The buffer 312 receives gradients 208-(i+1) from the next layer 204-(i+1), and sends the gradients 208-(i+1) to P backward path PEs 310-0 through 310-(P-1). The backward path PEs 310-0 through 310-(P-1) may perform a gradient generation function to generate gradients 208-i based on the activations 206-(i-1) from the preceding layer 204-(i-1) and the gradients 208-(i+1) from the next layer 204-(i+1). The gradients 208-i may be sent to the preceding layer 204-(i-1), e.g., through a buffer 314. In some embodiments, each backward path PE corresponds to a neuron of the layer 204-i. As such, the number P of backward path PEs in the backward path unit 302 may control the number of neurons of the layer 204-i that may be computed in parallel in the backward path process. In some embodiments, the backward path PEs 310-0 through 310-(P-1) may receive activation inputs 316 from corresponding forward path PEs 306-0 through 306-(P-1) respectively, and perform the gradient generation function based on the activations 206-(i-1) from the preceding layer 204-(i-1), the gradients 208-(i+1) from the next layer 204-(i+1), and the activation inputs 316 from a corresponding forward path PE.

[0051] Referring to FIG. 4, illustrated is an exemplary forward path PE 306-p of a forward path unit 300 of a layer 204-i. The forward path PE 306-p includes a weight and threshold memory 402, a multiply and accumulate unit 404, and an activation unit 406. The weight and threshold memory 402 may receive updated weights 318 (e.g., from a corresponding backward path PE 310-p of a backward path unit 302 of the layer 204-i), and send the weights 318 to the multiply and accumulate unit 404. The multiply and accumulate unit 404 may receive activations 206-(i-1) from a preceding layer 204-(i-1), receive the weights 318 from the weight and threshold memory 402, and generates activations input 316 by performing a multiply and accumulate operation (e.g., by using a hardware multiplier-accumulator (MAC) unit). The activation unit input 316 may be sent to the activation unit 406, which generates activations 206-i based on the activation unit input 316 and the weights 318. The activations 206-i may then be provided to an input of the next layer 204-(i+1). In some embodiments, the activations input 316 may also be sent to the corresponding back-

ward path PE 310-p for computing gradients in the backward path process.

[0052] In the example of FIG. 4, the multiply and accumulate unit 404 may be scaled to compute S multiply accumulates per clock cycle for a single dot product. In the forward path unit 300 of FIG. 3, the number P of forward path PEs controls the number of dot products computed in parallel. As a result, a single layer 204-i may compute S synapses of P neurons each clock cycle in the forward path process.

[0053] Referring to FIG. 5, illustrated is an exemplary backward path PE 310-p of a backward path unit 302 of a layer 204-i. The backward path PE 310-p includes an activation backpropagation unit 502. The activation backpropagation unit 502 receives gradients 208-(i+1) (e.g., from the next layer 204-(i+1) or the loss computation unit 210) and activations input 316 from a corresponding forward path PE 306-p, and generates gradients 504 based on an activation gradient function. The multiply and accumulate unit 506 may generate gradients 208-i based on weights 318 (e.g. received from the weight and threshold memory 514) and the gradients 504. The gradients 208-i may then be provided to a preceding layer 204-(i-1).

[0054] In the example of FIG. 5, the multiply and accumulate unit 506 may be scaled to compute S multiply accumulates per clock cycle for a single dot product. In the backward path unit 302 of FIG. 3, the number P of backward path PEs controls the number of dot products computed in parallel. As a result, a single layer 204-i may compute S synapses of P neurons each clock cycle in the backward path process.

[0055] In the example of FIG. 5, the gradients 504 may also be sent to an optimization unit 508. The optimization unit 508 may update weights and thresholds based on the gradients 504 and the activations 206-(i-1) (e.g., from the preceding layer 204-(i-1) or the preprocessing unit 202, and send the updated weights 510 and thresholds 512 to a weight and threshold memory 514. The optimization unit 508 may use optimization algorithms including, for example, stochastic gradient descent, batch gradient descent, gradient descent, Adam, and/or any other applicable optimization algorithms. In some embodiments, the weight and threshold memory 514 may send the weights and thresholds 318 to a corresponding forward path PE 306-p.

[0056] The training algorithm (e.g., a backpropagation algorithm) used for training of the neural network 200 may have various dependencies. Such dependencies may result in stalling and cause difficulties for implementing the neural network using multiple accelerators. For example, in a backpropagation algorithm, during the forward path process, a particular layer 204-i may require that its weights for the batch b , denoted as $W_i(b)$, have been calculated in the backward path process before its forward path unit 300 computes activations for the batch b , denoted as $Act_i(b)$. Such a dependency between the forward path and backward path may prevent using multiple compute units (e.g., accelerators) for different layers

204-0 through 204-(N-1). For example, because of the dependency, when multiple compute units are used to implement different groups of layers, those compute units may be constantly waiting for either new activation inputs in the forward path process or new gradient inputs for the backward path process.

[0057] To address the dependency of the backpropagation algorithm, a single compute unit (e.g., a single GPU or a single systolic array) may be used to implement the neural network training. In such embodiments, the single computing unit computes the forward path process layer by layer (e.g., from layer 204-0 through layer 204-(N-1)), and then computes the backward path layer by layer (e.g., from layer 204-(N-1) through layer 204-0) in a reverse order. However, in some embodiments, layers (e.g., layers in Convolutional Neural Networks (CNNs)) may have significantly different computation and memory requirements. As such, a "one-size-fits-all" solution that uses a single accelerator to implement all the layers often results in under-utilization of that accelerator when processing particular layers.

[0058] Referring to FIGS. 6, 7, 8, 9A, 9B, and 10, the neural network system 200 may use various techniques to remove the dependency in the backpropagation algorithm. For example, FIG. 6 illustrates a delayed model adaptation scheme that may be used to remove the dependencies between layers. For further example, FIGS. 9A and 9B illustrate a synthetic gradient estimator that may be used to provide gradient estimates, remove dependencies between layers, and reduce the delay factor of the layers. By removing the dependencies between the layers, the neural network may be implemented using multiple compute units (e.g., accelerators) assigned to process different layers. Such embodiments may reduce the training time of the neural network, and improve accelerator utilization by configuring accelerators based on different computation and memory requirements of the corresponding layers.

[0059] Referring to FIG. 6, in some embodiments, a neural network system 600 may use a delayed model adaptation scheme to remove the dependencies between the layers, thereby enabling the efficient usage of multiple accelerators (e.g., multiple GPUs, multiple FPGAs, a single FPGA including multiple systolic arrays) for implementing the neural network system 200. The neural network system 600 of FIG. 6 is substantially similar to the neural network system 200 of FIG. 2 except the differences described below. By using the delayed model adaptation, in a layer 204-i, activation outputs 206-i, denoted as $Act_i(b)$ for a batch having an index b (also referred to as the b^{th} batch) of the preprocessed training set 222, may be computed using gradients 208-(i+1) for the $(b-d)^{\text{th}}$ batch, denoted as $Grad_{i+1}(b-d)$. For example, activation outputs 206-i $Act_i(b)$ may be computed by using weights $W_i(b-d)$ for the $(b-d)^{\text{th}}$ batch, where weights $W_i(b-d)$ may be computed using $Grad_{i+1}(b-d)$, which may then be used to compute activation outputs 206-i $Act_i(b)$. This allows $d-1$ periods of delay between the calculation

of the activation function (e.g., performed by the activation unit 406 of a forward path PE 306-p in a layer 204-i) and the weight update (e.g., performed by the optimization unit 508 of a backward path PE 310-p).

[0060] As shown in FIG. 6, at a particular time t_0 , each layer operates on a different batch of the preprocessed training set 222. As such, the layers may have different delay factors (latency) d . For example, at time t_0 , layer 204-1 receives $Act_0(b-1)$ and $Grad_2(b-2N-1)$, and performs computation to generate $Act_1(b-1)$ based on $Act_0(b-1)$ and $Grad_2(b-2N-1)$. As such, layer 204-1 has a delay factor d_1 of $2N$. For further example, at time t_0 , layer 204-2 receives $Act_1(b-2)$ and $Grad_3(b-2N)$, and performs the computation to generate $Act_2(b-2)$ using $Act_1(b-2)$ and $Grad_3(b-2N)$. As such, layer 204-2 has a delay factor d_2 of $2N-1$. In the example of FIG. 6, the delay factor increase as the layer index increases. In other words, layers near layer 0 have delay factors that are greater than the delay factors of the layers near layer $N-1$.

[0061] Each of the layers 204-0 through 204-($N-1$) may adjust its learning rate μ and/or step size based on its delay factor d . Accordingly, the layers 204-0 through 204-($N-1$) may have different learning rates. In an example, a layer 204-i may have a learning rate lower than a layer 204-j following the layer 204-i, where the delay factor d_i of the layer 204-i is greater than the delay factor d_j of the layer 204-j. Extra hardware may be used to calculate correction terms that may be used to ensure the stability of the parameters learned using the delayed model adaptation and compensate for any error introduced by inclusion of the delay model adaptation.

[0062] Note that both neural network systems 200 and 600 may utilize the same types of processing engines, but as discussed above, the ways that these processing engines are scheduled to perform the forward path process and backward path process vary significantly.

[0063] Referring to FIG. 7, illustrated is a portion of a layer 204-i implementing delayed model adaptation. In the example of FIG. 7, at a particular time t_0 , a weight gradient function unit 702 receives gradients $208-(i+1)$ $Grad_{i+1}(b-d)$ for the $(b-d)^{th}$ batch from the next layer 204($i+1$). The weight gradient function unit 702 also receives delayed activation 704 from a delay unit 706. The delay unit 704 may generate the delayed activation 704 by applying d periods of delay (e.g., associated with d batches) to activations $206-(i-1)$ $Act_{i-1}(b)$ for the b^{th} batch. The weight gradient function unit 702 outputs weights 708. The weights 708 are optimized by the optimization function 710. An additional delay (e.g., one period of delay) may be applied to the resulting optimized weights by a delay unit 712 to generate weights 714 (e.g., weights 318 of FIG. 4). The weights 714 may be stored in a memory 402.

[0064] In the forward path 300, the weights 714 are used to generate activations $206-i$ $Act_i(b)$ for the b^{th} batch. As illustrated in the example of FIG. 7, the weights 714 are sent to a neuron input function unit 716 (e.g. multiply and accumulate unit 404 of FIG. 4), which per-

forms neuron input functions to activations $206-(i-1)$ $Act_{i-1}(b)$ using weights 714. The neuron input function unit 716 may send activation unit input 718 (e.g., activation unit input 316) to the activation unit 720 (e.g., activation unit 406 of FIG. 4). The activation unit 720 generates activations $206-i$ $Act_i(b)$ for the b^{th} batch based on the activation unit input 718. The activations $206-i$ $Act_i(b)$ may then be provided to an input of the next layer 204-($i+1$).

[0065] In the backward path 302, the weights 714 are used to generate gradients $208-i$ $Grad_i(b-d)$. As illustrated in the example of FIG. 7, a weight gradient function unit 702 (e.g., activation backpropagation unit 502 of FIG. 5) receives gradients 722. The gradients 722 and weights 714 are sent to a neuron gradient function unit 724 (e.g., a multiply and accumulate unit 506 of FIG. 5). The neuron gradient function unit 724 may generate gradients $208-i$ $Grad_i(b-d)$ for the $(b-d)^{th}$ batch, and send the gradients $Grad_i(b-d)$ to the previous layer 204-($i-1$).

[0066] Referring to FIG. 8, illustrated is a method 800 for implementing the delayed model adaptation for training a neural network using a backpropagation algorithm. The method begins at block 802, where a layer 204-i of a neural network receives activations $Act_{i-1}(b-d)$ from layer 204-($i-1$) and gradients $Grad_{i+1}(b-d)$ from layer 204-($i+1$). The method 800 may proceed to block 804, where weights $W_i(b-d)$ are generated based on $Act_{i-1}(b-d)$ and $Grad_{i+1}(b-d)$. For example, the backward path PE 310-p may generate $W_i(b-d)$ as described above with reference to FIG. 7. The method 800 may then proceed to block 806, where weights $W_i(b-d)$ may be used to generate activations for batch t of the training set in the forward path at the layer 204-i. For example, weights $W_i(b-d)$ may be sent to the corresponding forward path PE 306-p of the layer 204-i. The forward path PE 306-p may then use weights $W_i(b-d)$ to compute $Act_i(b)$ as discussed above with reference to FIGS. 3 and 4. As such, by using the delayed model adaptation, dependencies between layers for the forward path process and the backward path process are removed.

[0067] Referring to FIGS. 9A and 9B, synthetic gradients may be used to remove the dependency in the backpropagation algorithm. In the example of FIG. 9A, a portion of a backward path PE 310-p of a layer 204-i employing synthetic gradients is illustrated. The backward path PE 310-p includes a synthetic gradient estimator 950 including a synthetic gradient generator 902. The synthetic gradient generator 902 receives activations $206-(i-1)$ from a preceding layer 204-($i-1$), and generates gradient estimates 904 based on the activations $206-(i-1)$ and a gradient estimate error 908. The gradient estimates 904 may then be sent to the activation backpropagation units 502 of the backward path PE 310-p. Weights 318 may be computed based on the gradient estimates 904, and be sent to the corresponding forward path PE 306-p for generating activations. As such, by using synthetic gradients, dependencies between layers for the forward path process and the backward path proc-

ess are removed.

[0068] In the example of FIG. 9A, the gradient estimates 904 may be sent to a gradient estimate error generator 906 (e.g., a subtractor). After the actual gradients 208-(i+1) becomes available (e.g., from a layer 204-(i+1)), the gradient estimate error generator 906 may compare the gradient estimate 904 and actual gradients 208-(i+1) to generate a gradient estimate error 908. The synthetic gradient generator 902 may improve future gradient estimates 904 based on the gradient estimate error 908. For example, the gradient estimate error 908 may be used to generate a training example for the synthetic gradient generator 902 to perform machine learning based on a machine learning algorithm (e.g., a neural network, least mean squares (LMS)). In a particular example, by using a simple LMS learning algorithm for machine learning, the hardware costs of the synthetic gradient generator 902 may be reduced. In some examples, the synthetic gradients may provide gradient estimates without actual gradients from the next layer.

[0069] Referring to FIG. 9B, a synthetic gradient generator may be used together with delayed model adaptation to remove the dependency between forward and backward paths and reduce the amount of delay incurred in early hidden layers. In the example of FIG. 9B, a synthetic gradient estimator 950 may receive, from the layer 204-i, activations 206-i $Act_i(b_{i+1})$ for the $(b_{i+1})^{th}$ batch. The synthetic gradient estimator 950 may receive, from the following layer 204-(i+1), gradients $\tilde{Grad}_{i+1}(b - d_2)$ for the $(b - d_2)^{th}$ batch. The synthetic gradient estimator 950 may generate gradients $\tilde{Grad}_{i+1}(b - d_1)$ for the $(b - d_1)^{th}$ batch based on the activations 206-i $Act_i(b_{i+1})$ and the gradients $\tilde{Grad}_{i+1}(b - d_2)$. In an example, d_1 is less than d_2 . By using gradients $\tilde{Grad}_{i+1}(b - d_2)$, the $\tilde{Grad}_{i+1}(b - d_1)$ may be generated without waiting for the layer 204-(i+1) to complete processing the $(b - d_1)^{th}$ batch, which may reduce the delay factor in the layers.

[0070] Referring to FIG. 10, the performance for training the neural network systems 200 and 600 may be improved by using reduced precision networks (e.g., a binarized neural network) and custom data presentations (e.g., logarithmic data representations) to improve training efficiency. For example, when training a binarized neural network, during the forward path process, the multiplications may be performed using exclusive-NOR circuits (XNORs), and the accumulation may be performed using bit counts. For further example, logarithmic data representations may bring various benefits to hardware implementations including, for example, multiplies in the training algorithm may be implemented as additions in the hardware, and additions/subtractions in the training algorithm may be implemented using table lookups.

[0071] The example of FIG. 10 is a portion of a backward path PE 310-p of a layer 204-i implementing a binarized neural network. In the example of FIG. 10, during the backward path process, multiplies may only be performed between 1s and -1s. For fixed-point data types, multiplies may be either a unity function or an element-

wise bit-flip plus a 1-bit value. For logarithmic data types (including e.g., a float data type), multiplies become a simple bit-flip of the sign bit. Furthermore, particular activation gradient functions (e.g. a hardtanh activation function) may be used such that the output 1008 is either 0 or 1. An XNOR circuit 1002 may receive gradients 208-(i+1) that are binary values and the output 1008, and generates gradients 504 which may be provided to the optimization unit 508 of the backward path PE 310-p. In the example of FIG. 10, a multiply and accumulate unit 506 includes an XNOR circuit 1004 controlled by binary weights 318 and an adder 1006, and may generate gradients 208-i that may be provided to the preceding layer 204-(i-1). As such, by using reduced precision data type and logarithmic data presentation, the backward path PE 310-p may perform the backward path process without using any multipliers.

[0072] Referring to FIGS. 11, 12, and 13, various architectures may be used to implement the neural network systems 200 and 600 for scaling the neural network systems to large problems. The example of FIG. 11 illustrates a streaming architecture that uses multiple programmable ICs (e.g., multiple FPGAs) and their streaming interfaces for communication between those programmable ICs. The example of FIG. 12 illustrates an architecture using a single programmable IC (e.g., a single FPGA) including a single programmable systolic array. The example of FIG. 13 uses a single programmable IC (e.g., a single FPGA) including two systolic arrays.

[0073] Referring to FIG. 11, a streaming architecture 1100 may be implemented using multiple ICs (e.g., programmable ICs each on a separate chip) using streaming interfaces between them. The streaming architecture 1100 includes a neural network system 1102 includes L ICs (e.g., FPGAs, GPUs) 1106-0 through 1106-(L-1), where L is a positive integer. Each of the IC 1106-i may be used to implement one or more layers. In the example of FIG. 11, the IC 1106-0 is used to implement layers 204-0 through 204-2, and the IC 1106-(L-1) is used to implement layers 204-(N-3), 204-(N-2), and 204-(N-1). A host 1104 may send a training set 220 to the neural network system 1102, and may receive learned weights from the neural network system 1102.

[0074] Referring to FIG. 12, a single IC 1202 including a single processing array 1204 and an IO buffer 1206 may be used to implement the neural network system 200. The single processing array 1204 may be a systolic array, and may be referred to as a systolic array 1204. The systolic array 1204 may be used to compute the forward path and/or backward path for fully connected layers, convolutional layers, and/or max-pooling layers of the neural network system 200. In an example, the layers 204-0 through 204-(N-1) are time-multiplexed onto the systolic array 1204. In other words, the systolic array 1204 may be configured to implement the layer 204-0, then the layer 204-1, and so on until implementing the layer 204-(N-1). The results generated by the systolic array 1204 for a layer 204-i may be saved to be used as

input to the next implementation of a layer (e.g., a next layer 204-(i+1) in the forward path process, and a preceding layer 204-(i-1) in the backward path process).

[0075] Referring to FIG. 13, a single programmable IC 1302 including two systolic arrays 1304 and 1306 and I/O buffers 1308 are used. For example, the systolic array 1304 may be used to compute the forward path process for all the layers 204-0 through 204-(N-1), and is also referred to as the forward path systolic array 1304. For further example, the systolic array 1306 may be used to compute the backward path process for all the layers 204-(N-1) through 204-0, and may also be referred to as the backward path systolic array 1306. The systolic arrays 1304 and 1306 may be scheduled to minimize the adaptation delay in all layers 204-0 through 204-(N-1).

[0076] To train the neural network systems 200 and 600, a user may generate descriptions for the neural network. In some examples, based on the description, custom hardware may be generated. In some examples, a bitstream for an accelerator corresponding to a predetermined architecture (e.g., the architectures of FIGS. 11, 12, and/or 13) may be loaded based on the description. The choice of whether to generate a custom architecture may be determined based on performance models generated architectures, the target device, place and route time estimates, the dataset size, the number of iterations. The place and route time estimates, the dataset size, and the number of iterations may be generated offline. In some embodiments, the systolic arrays may be scheduled to minimize the adaptation delay in all layers.

[0077] It is noted that various configurations (e.g., the number N of layers, the number S of multiply accumulates computed by the multiply and accumulate unit each clock cycle, the number P of forward/backward path PEs) illustrated in FIGS. 2-13 are exemplary only and not intended to be limiting beyond what is specifically recited in the claims that follow. It will be understood by those skilled in that art that other configurations may be used. A delayed model adaptation scheme of FIGS. 6 through 8 may be used together with synthetic gradients described with reference to FIG. 9. In other embodiments, the delayed model adaptation scheme may be used to replace synthetic gradients entirely to reduce the hardware cost of synthetic gradients. In some embodiments, one or more optimization techniques (e.g., reduced precision networks, logarithmic data representations, other applicable optimization techniques, or a combination thereof) may be used in the architectures of FIGS. 11, 12, and 13. Furthermore, while binary values are used as an exemplary representation, various representations including for example ternary values, fixed point, floating point, and log2 representation may be used.

Claims

1. A neural network system (200), comprising:
one or more hardware processors configured to ex-

ecute instructions to cause the system to implement a neural network model, wherein the implementation of the neural network model comprises:
forward path processing elements (306-0...306(P-1)) and backward path processing elements (310-0...310-(P-1)),

wherein each neuron of the neural network model has a corresponding forward path processing element (306-0...306(P-1)) and a corresponding backward path processing element (310-0...310-(P-1)),

wherein the system (200) is further configured for:

implementing an input neural network layer (204-0) configured to:

receive a first training set (220) including a sequence of batches; and

provide, to a following neural network layer, a plurality of output activations (206-1 - 206-(N-1)) associated with the sequence of batches in a forward path, respectively, wherein the forward path in the following neural network layer includes the one or more forward path processing elements (306-0...306(P-1)), wherein the plurality of output activations (206-1...206(N-1)) are computed using the one or more forward path processing elements (306-0...306(P-1));

implementing first and second hidden neural network layers (204-1; 204-2) following the input neural network layer (204-0), each neural network hidden layer configured to:

receive, from a preceding neural network layer, a first input activation associated with a first batch of the sequence of batches;

perform, using the forward path processing elements (306-0, 306-(P-1)), an activation generation function to generate activations (206-i) based on activations (204-i) received from the forward path processing elements (306-0, 306-(P-1)) corresponding to the neurons of the preceding layer (204-(i-1)), and send the generated activations (206-i) to the forward path processing elements (306-0, 306-(P-1)) corresponding to the neurons of the next layer (204-(i+1)),

receive, when the first input activation is received, a first input gradient (208-(i+1)) computed in the one or more backward path processing elements (310-0...310-(P-1)) of a subsequent

neural network layer and associated with a second batch of the sequence of batches preceding the first batch by a delay factor (d_i) associated with the first batch, the calculations in the one or more backward path processing elements (310-0...310-(P-1)) in each hidden neural network layer being implemented in parallel to the calculations in the one or more forward path processing elements, ;
 wherein the delay factor (d_i) corresponds to a latency of two or more batches other than the first and second batches; and
 perform, using the backward path processing elements (310-0, 310-(P-1)), a gradient generation function to generate gradients (208-i) based on the received first input activation (206-(i-1)) and on the received first input gradient (208-(i+1));
 provide, to the preceding neural network layer, the generated gradients (208-i);
 provide, to a following neural network layer of the hidden neural network layer, a first output activation associated with the first batch, wherein the first output activation is determined based on the first input activation and the first input gradient;
 wherein the first hidden neural network layer (204-1) has a first delay factor that is different from a second delay factor of the second hidden layer;
 a neural network output layer (204-(N-1)) following the first and second hidden neural network layers, wherein the output layer (204-(N-1)) is configured to:

receive, from a preceding neural network layer, a second input activation; and
 provide, to the preceding neural network layer, a first output gradient (208-N - 208-1) based on the second input activation and the first training set (220), wherein the first output gradient (208-N...208-1) is computed using the one or more backward path processing elements (310-0...310-(P-1)).

2. The neural network system (200) of claim 1, wherein the first hidden layer (204-1) is configured to:
 provide, to the preceding neural network layer, a second output gradient associated with the second

batch of the sequence of batches based on the first input activation and the first input gradient, wherein the second output gradient (208-N...208-1) is computed using the one or more backward path processing elements (310-0...310-(P-1)).

3. The neural network system (200) of claim 2, wherein the first hidden neural network layer (204-1) is configured to:

generate a first weight associated with the second batch of the sequence of batches based on a third input activation associated with the second batch of the sequence of batches and the first input gradient; and
 generate the first output activation associated with the first batch of the sequence of batches based on the first input activation, the first input gradient, and the first weight.

4. The neural network system (200) of claim 3, wherein the first hidden neural network layer (204-1) is configured to:

generate the second output gradient based on the first input activation, first input gradient, and first weight.

5. The neural network system (200) of claim 3, wherein the first hidden neural network layer (204-1) is configured to:

adjust a learning rate based on the delay factor between the first batch of the sequence of batches and the second batch of the sequence of batches; and
 generate the first weight associated with the second batch of the sequence of batches by applying the learning rate to the first input gradient associated with the second batch of the sequence of batches.

6. The neural network system (200) of claim 1, further comprising:
 a preprocessing unit (202) configured to:

receive an original training set (220) having a first size; and
 generate the first training set based on the original training set (220), wherein the first training set has a second size greater than the first size.

7. The neural network system (200) of claim 1, wherein the first hidden neural network layer (204-1) is coupled to a synthetic gradient generator (902) that is configured to:

generate a first input gradient estimate using a second input gradient associated with a third

- batch preceding the second batch of the sequence of batches; and
provide to the first hidden layer (204-1) the first input gradient estimate as the first input gradient.
8. The neural network system (200) of claim 7, wherein the synthetic gradient generator (902) is configured to:
- receive a second input gradient from the following neural network layer;
 - generate a gradient estimate error based on the first input gradient estimate and the second input gradient; and
 - generate a second input gradient estimate based on the gradient estimate error using a machine learning algorithm.
9. The neural network system (200) of claim 1, wherein the first input activation, the first output activation, the second input activation, and the first output gradient include binary values.
10. The neural network system (200) of claim 1, further comprising:
- a logarithmic data representation configured to quantize the first input activation, the first output activation, the second input activation, and the first output gradient in a log-domain.
11. The neural network system (200) of claim 1, further comprising:
- a plurality of accelerators,
 - wherein each accelerator is configured to implement one layer of the input layer, the one or more hidden layers, and the output layer (204-(N-1)), and
 - wherein each accelerator is configured to have a computational capacity and a memory capacity based on the respective layer.
12. A computer-implemented method of implementing a neural network model, wherein the implementation of the neural network model comprises forward path processing elements (306-0...306(P-1)) and backward path processing elements (310-0...310-(P-1)), wherein each neuron of the neural network model has a corresponding forward path processing element (306-0...306(P-1)) and a corresponding backward path processing element (310-0...310-(P-1)), wherein the computer-implemented method comprises:
- receiving, by an input neural network layer (204-0) of a neural network, a first training set (220) including a sequence of batches;
 - providing, by the input neural network layer

(204-0) to a following neural network layer, a plurality of output activations associated with the sequence of batches in a forward path, respectively, wherein the forward path in the following neural network layer includes the one or more forward path processing elements (306-0...306(P-1)), wherein the plurality of output activations (206-1...206(N-1)) are computed using the one or more forward path processing elements (306-0...306(P-1));

receiving, by a first hidden neural network layer (204-1) of the neural network from a preceding neural network layer, a first input activation associated with a first batch of the sequence of batches;

performing, using the forward path processing elements (306-0, 306-(P-1)), an activation generation function to generate activations (206-i) based on activations (204-i) received from the forward path processing elements (306-0, 306-(P-1)) corresponding to the neurons of the preceding layer (204-(i-1)), and send the generated activations (206-i) to the forward path processing elements (306-0, 306-(P-1)) corresponding to the neurons of the next layer (204-(i+1),

receiving, by the first hidden neural network layer (204-1) when the first input activation is received, a first input gradient (208-(i+1)) computed in the one or more backward path processing elements (310-0...310-(P-1)) of a subsequent neural network layer and associated with a second batch of the sequence of batches preceding the first batch by a delay factor (d_i) associated with the first batch, wherein the calculations in the one or more backward path processing elements (310-0...310-(P-1)) in each hidden layer are implemented in parallel to the calculations in the one or more forward path processing elements and the delay factor (d_i) corresponds to a latency of two or more batches other than first and second batches of the sequence of batches;

performing, using the backward path processing elements (310-0, 310-(P-1)), a gradient generation function to generate gradients (208-i) based on the received first input activation (206-(i-1)) and on the received first input gradient (208-(i+1));

providing, to the preceding neural network layer, the generated gradients (208-i);

providing, by the first hidden neural network layer (203-1) to a second hidden neural network layer (204-2) following the first hidden layer (204-1), a first output activation associated with the first batch, wherein the first output activation is based on the first input activation and the first input gradient;

wherein the first hidden neural network layer

(204-1) has a first delay factor that is different from a second delay factor of the second hidden neural network layer;
 receiving, by an output neural network layer (204-(N-1)) following the first hidden neural network layer (204-1) from a preceding neural network layer of the output neural network layer (204-(N-1)), a second input activation; and
 providing, by the output neural network layer (204-(N-1)) to the preceding neural network layer, a first output gradient based on the second input activation and the first training set (220), wherein the first output gradient (208-N...208-1) is computed using the one or more backward path processing elements (310-0...310-(P-1)).

13. The method of claim 12, further comprising:
 providing, by the first hidden neural network layer (204-1) to the preceding neural network layer, a second output gradient associated with the second batch of the sequence of batches based on the first input activation and the first input gradient.

14. The method of claim 13, further comprising:
 generating, by the first hidden neural network layer (204-1), a first weight associated with the second batch of the sequence of batches based on a third input activation associated with the second batch and the first input gradient; and
 generating, by the first hidden neural network layer (204-1), the first output activation associated with the first batch of the sequence of batches based on the first input activation, the first input gradient, and the first weight.

15. The method of claim 14, further comprising:
 performing at least one operation selected from:
 generating, by the first hidden neural network layer (204-1), the second output gradient based on the first input activation, the first input gradient, and the first weight; and
 adjusting, by the first hidden neural network layer (204-1), a learning rate based on the delay factor between the first batch and second batch and generating, by the first hidden neural network layer (204-1), the first weight associated with the second batch of the sequence of batches by applying the learning rate to the first input gradient associated with the second batch.

Patentansprüche

1. System (200) eines neuronalen Netzwerks, umfassend:
 einen oder mehrere Hardwareprozessoren, die kon-

figuriert sind, um Anweisungen auszuführen, die das System veranlassen, ein Modell des neuronalen Netzwerks zu implementieren, wobei die Implementierung des Modells des neuronalen Netzwerks umfasst:

Vorwärtspfadverarbeitungselemente (306-0...306-(P-1)) und Rückwärtspfadverarbeitungselemente (310-0...310-(P-1)), wobei jedes Neuron des Modells des neuronalen Netzwerks ein entsprechendes Vorwärtspfadverarbeitungselement (306-0...306-(P-1)) und ein entsprechendes Rückwärtspfadverarbeitungselement (310-0...310-(P-1)) aufweist, wobei das System (200) ferner konfiguriert ist zum:

Implementieren einer Eingabeschicht (204-0) des neuronalen Netzwerks, die konfiguriert ist zum:

Empfangen eines ersten Trainingssatzes (220), der eine Folge von Stapeln einschließt; und

Bereitstellen, an eine folgende Schicht des neuronalen Netzwerks, einer Vielzahl von Ausgabeaktivierungen (206-1 - 206-(N-1)), die jeweils mit der Folge von Stapeln in einem Vorwärtspfad verknüpft ist, wobei der Vorwärtspfad in der folgenden Schicht des neuronalen Netzwerks die ein oder mehrere Vorwärtspfadverarbeitungselemente (306-0...306-(P-1)) einschließt, wobei die Vielzahl von Ausgabeaktivierungen (206-1...206-(N-1)) unter Verwendung der ein oder mehreren Vorwärtspfadverarbeitungselemente (306-0...306-(P-1)) berechnet wird;

Implementieren einer ersten und einer zweiten verborgenen Schicht (204-1; 204-2) des neuronalen Netzwerks im Anschluss an die Eingabeschicht (204-0) des neuronalen Netzwerks, wobei jede verborgene Schicht des neuronalen Netzwerks konfiguriert ist zum:

Empfangen, von einer vorstehenden Schicht des neuronalen Netzwerks, einer ersten Eingabeaktivierung, die mit einem ersten Stapel der Folge von Stapeln verknüpft ist;

Durchführen, unter Verwendung der Vorwärtspfadverarbeitungselemente (306-0, 306-(P-1)), einer Aktivierungserzeugungsfunktion, um Aktivierungen (206-i) basierend auf Aktivierungen (204-i) zu erzeugen, die von den Vorwärtspfadverarbeitungselementen (306-0, 306-(P-1)) empfangen werden,

die den Neuronen der vorstehenden Schicht (204-(i-1)) entsprechen, und Senden der erzeugten Aktivierungen (206-i) an die Vorwärtspfadverarbeitungselemente (306-0, 306-(P-1)), die den Neuronen der nächsten Schicht (204-(i+1)) entsprechen, 5

Empfangen, wenn die erste Eingabeaktivierung empfangen wird, eines ersten Eingabegradienten (208-(i+1)), der in dem einen oder den mehreren Rückwärtspfadverarbeitungselementen (310-0...310-(P-1)) einer nachfolgenden Schicht des neuronalen Netzwerks berechnet wird und mit einem zweiten Stapel der Folge von Stapeln verknüpft ist, der dem ersten Stapel um einen Verzögerungsfaktor (d_i) vorsteht, der mit dem ersten Stapel verknüpft ist, wobei die Berechnungen in dem einen oder den mehreren Rückwärtspfadverarbeitungselementen (310-0...310-(P-1)) in jeder verborgenen Schicht des neuronalen Netzwerks parallel zu den Berechnungen in dem einen oder den mehreren Vorwärtspfadverarbeitungselementen implementiert werden; wobei der Verzögerungsfaktor (d_i) einer Latenz von zwei oder mehr Stapeln entspricht, die nicht der erste und der zweite Stapel sind; und 10

Durchführen, unter Verwendung der Rückwärtspfadverarbeitungselemente (310-0, 310-(P-1)), einer Gradientenerzeugungsfunktion, um Gradienten (208-i) basierend auf der empfangenen ersten Eingabeaktivierung (206-(i-1)) und auf dem empfangenen ersten Eingabegradienten (208-(i+1)) zu erzeugen; 15

Bereitstellen, an die vorstehende Schicht des neuronalen Netzwerks, der erzeugten Gradienten (208-i); 20

Bereitstellen, an eine folgende Schicht des neuronalen Netzwerks der verborgenen Schicht des neuronalen Netzwerks, eine erste Ausgabeaktivierung, die mit dem ersten Stapel verknüpft ist, wobei die erste Ausgabeaktivierung basierend auf der ersten Eingabeaktivierung und dem ersten Eingabegradienten bestimmt wird; 25

wobei die erste verborgene Schicht (204-1) des neuronalen Netzwerks einen ersten Verzögerungsfaktor aufweist, der sich von einem zweiten Verzögerungsfaktor der zweiten verborgenen Schicht unterscheidet; 30

eine Ausgabeschicht (204-(N-1)) des neuronalen Netzwerks, die der ersten und der zweiten verborgenen Schicht des neuronalen Netzwerks folgt, wobei die Ausgabeschicht (204-(N-1)) konfiguriert ist zum:

Empfangen, von einer vorstehenden Schicht des neuronalen Netzwerks, einer zweiten Eingabeaktivierung; und 35

Bereitstellen, an die vorstehende Schicht des neuronalen Netzwerks, eines ersten Ausgabegradienten (208-N - 208-1) basierend auf der zweiten Eingabeaktivierung und dem ersten Trainingssatz (220), wobei der erste Ausgabegradient (208-N...208-1) unter Verwendung des einen oder der mehreren Rückwärtspfadverarbeitungselemente (310-0...310-(P-1)) berechnet wird. 40

2. System (200) des neuronalen Netzwerks nach Anspruch 1, wobei die verborgene Schicht (204-1) konfiguriert ist zum:

Bereitstellen, an die vorstehende Schicht des neuronalen Netzwerks, eines zweiten Ausgabegradienten, der mit dem zweiten Stapel der Folge von Stapeln verknüpft ist, basierend auf der ersten Eingabeaktivierung und dem ersten Eingabegradienten, wobei der zweite Ausgabegradient (208-N...208-1) unter Verwendung des einen oder der mehreren Rückwärtspfadverarbeitungselemente (310-0...310-(P-1)) berechnet wird. 45

3. System (200) des neuronalen Netzwerks nach Anspruch 2, wobei die erste verborgene Schicht (204-1) des neuronalen Netzwerks konfiguriert ist zum:

Erzeugen einer ersten Wertigkeit, die mit dem zweiten Stapel der Folge von Stapeln verknüpft ist, basierend auf einer dritten Eingabeaktivierung, die mit dem zweiten Stapel der Folge von Stapeln und dem ersten Eingabegradienten verknüpft ist; und 50

Erzeugen der ersten Ausgabeaktivierung, die mit dem ersten Stapel der Folge von Stapeln verknüpft ist, basierend auf der ersten Eingabeaktivierung, dem ersten Eingabegradienten und der ersten Wertigkeit. 55

4. System (200) des neuronalen Netzwerks nach Anspruch 3, wobei die erste verborgene Schicht (204-1) des neuronalen Netzwerks konfiguriert ist zum:

Erzeugen des zweiten Ausgabegradienten basierend auf der ersten Eingabeaktivierung, dem ersten 60

Eingabegradien und der ersten Wertigkeit.

5. System (200) des neuronalen Netzwerks nach Anspruch 3, wobei die erste verborgene Schicht (204-1) des neuronalen Netzwerks konfiguriert ist zum:

Anpassen einer Lernrate basierend auf dem Verzögerungsfaktor zwischen dem ersten Stapel der Folge von Stapeln und dem zweiten Stapel der Folge von Stapeln; und Erzeugen der ersten Wertigkeit, die mit dem zweiten Stapel der Folge von Stapeln verknüpft ist, durch Anwenden der Lernrate auf den ersten Eingabegradien, der mit dem zweiten Stapel der Folge von Stapeln verknüpft ist.

6. System (200) des neuronalen Netzwerks nach Anspruch 1, ferner umfassend: eine Vorverarbeitungseinheit (202), die konfiguriert ist zum:

Empfangen eines ursprünglichen Trainingssatzes (220), der eine erste Größe aufweist; und Erzeugen des ersten Trainingssatzes basierend auf dem ursprünglichen Trainingssatz (220), wobei der erste Trainingssatz eine zweite Größe aufweist, die größer als die erste Größe ist.

7. System (200) des neuronalen Netzwerks nach Anspruch 1, wobei die erste verborgene Schicht (204-1) des neuronalen Netzwerks mit einem Erzeuger (902) eines synthetischen Gradienten gekoppelt ist, der konfiguriert ist zum:

Erzeugen einer ersten Eingabegradien-schätzung unter Verwendung eines zweiten Eingabegradien, der mit einem dritten Stapel verknüpft ist, der dem zweiten Stapel der Folge von Stapeln vorsteht; und Bereitstellen der ersten verborgenen Schicht (204-1) die erste Eingabegradien-schätzung als ersten Eingabegradien.

8. System (200) des neuronalen Netzwerks nach Anspruch 7, wobei der Erzeuger (902) des synthetischen Gradienten konfiguriert ist zum:

Empfangen eines zweiten Eingabegradien von der folgenden Schicht des neuronalen Netzwerks; Erzeugen eines Gradientenschätzfehlers basierend auf der ersten Eingabegradien-schätzung und dem zweiten Eingabegradien; und Erzeugen einer zweiten Eingabegradien-schätzung basierend auf dem Gradientenschätzfehler unter Verwendung eines maschinellen Lernalgorithmus.

9. System (200) des neuronalen Netzwerks nach Anspruch 1, wobei die erste Eingabeaktivierung, die erste Ausgabeaktivierung, die zweite Eingabeaktivierung und der erste Ausgabegradien binäre Werte einschließen.

10. System (200) des neuronalen Netzwerks nach Anspruch 1, ferner umfassend: eine logarithmische Datendarstellung, die konfiguriert ist, um die erste Eingabeaktivierung, die erste Ausgabeaktivierung, die zweite Eingabeaktivierung und den ersten Ausgabegradien in einer Log-Domäne zu quantisieren.

11. System (200) des neuronalen Netzwerks nach Anspruch 1, ferner umfassend:

eine Vielzahl von Beschleunigern, wobei jeder Beschleuniger konfiguriert ist, um eine Schicht der Eingabeschicht, der einen oder der mehreren verborgenen Schichten und der Ausgabeschicht (204-(N-1)) zu implementieren, und wobei jeder Beschleuniger konfiguriert ist, um eine Rechenkapazität und eine Speicherkapazität basierend auf der jeweiligen Schicht aufzuweisen.

12. Computerimplementiertes Verfahren zum Implementieren eines Modells des neuronalen Netzwerks, wobei die Implementierung des Modells des neuronalen Netzwerks Vorwärtspfadverarbeitungselemente (306-0... 306(P-1)) und Rückwärtspfadverarbeitungselemente (310-0...310-(P-1)) umfasst, wobei jedes Neuron des Modells des neuronalen Netzwerks ein entsprechendes Vorwärtspfadverarbeitungselement (306-0...306(P-1)) und ein entsprechendes Rückwärtspfadverarbeitungselement (310-0...310-(P-1)) aufweist, wobei das computerimplementierte Verfahren umfasst:

Empfangen, durch eine Eingabeschicht (204-0) eines neuronalen Netzwerks, eines ersten Trainingssatzes (220), der eine Folge von Stapeln einschließt; Bereitstellen, durch die Eingabeschicht (204-0) des neuronalen Netzwerks an eine folgende Schicht des neuronalen Netzwerks, einer Vielzahl von Ausgabeaktivierungen, die jeweils mit der Folge von Stapeln in einem Vorwärtspfad verknüpft ist, wobei der Vorwärtspfad in der folgenden Schicht des neuronalen Netzwerks das eine oder die mehreren Vorwärtspfadverarbeitungselemente (306-0...306(P-1)) einschließt, wobei die Vielzahl von Ausgabeaktivierungen (206-1...206(N-1)) unter Verwendung des einen oder der mehreren Vorwärtspfadverarbeitungselemente (306-0...306(P-1)) berechnet wird;

Empfangen, durch eine erste verborgene Schicht (204-1) des neuronalen Netzwerks des neuronalen Netzwerks von einer vorstehenden Schicht des neuronalen Netzwerks, einer ersten Eingabeaktivierung, die mit einem ersten Stapel der Folge von Stapeln verknüpft ist; 5

Durchführen, unter Verwendung der Vorwärtspfadverarbeitungselemente (306-0, 306-(P-1)), einer Aktivierungserzeugungsfunktion, um Aktivierungen (206-i) basierend auf Aktivierungen (204-i) zu erzeugen, die von den Vorwärtspfadverarbeitungselementen (306-0, 306-(P-1)) empfangen werden, die den Neuronen der vorstehenden Schicht (204-(i-1)) entsprechen, und 10

Senden der erzeugten Aktivierungen (206-i) an die Vorwärtspfadverarbeitungselemente (306-0, 306-(P-1)), die den Neuronen der nächsten Schicht (204-(i+1)) entsprechen, Empfangen, durch die erste verborgene Schicht (204-1) des neuronalen Netzwerks, wenn die 20

erste Eingabeaktivierung empfangen wird, eines ersten Eingabegradients (208-(i+1)), der in dem einen oder den mehreren Rückwärtspfadverarbeitungselementen (310-0...310-(P-1)) einer nachfolgenden Schicht des neuronalen Netzwerks berechnet wird, und mit einem zweiten Stapel der Folge von Stapeln verknüpft ist, 25

der dem ersten Stapel um einen Verzögerungsfaktor (d_i) vorsteht, der mit dem ersten Stapel verknüpft ist, wobei die Berechnungen in dem einen oder den mehreren Rückwärtspfadverarbeitungselementen (310-0...310-(P-1)) in jeder verborgenen Schicht parallel zu den Berechnungen in dem einen oder den mehreren Vorwärtspfadverarbeitungselementen implementiert werden und der Verzögerungsfaktor (d_i) einer Latenz von zwei oder mehreren Stapeln entspricht, die nicht der erste und der zweite Stapel der Folge von Stapeln sind; 30

Durchführen, unter Verwendung der Rückwärtspfadverarbeitungselemente (310-0, 310-(P-1)), einer Gradientenerzeugungsfunktion, um Gradienten (208-i) basierend auf der empfangenen ersten Eingabeaktivierung (206-(i-1)) und auf dem empfangenen ersten Eingabegradients (208-(i+1)) zu erzeugen; 40

Bereitstellen, an die vorstehende Schicht des neuronalen Netzwerks, der erzeugten Gradienten (208-i); 45

Bereitstellen, durch die erste verborgene Schicht (203-1) des neuronalen Netzwerks an eine zweite verborgene Schicht (204-2) des neuronalen Netzwerks, die der ersten verborgenen Schicht (204-1) folgt, einer ersten Ausgabeaktivierung, die mit dem ersten Stapel verknüpft ist, wobei die erste Ausgabeaktivierung auf der ersten Eingabeaktivierung und dem ersten Eingabegradients basiert; 50

55

wobei die erste verborgene Schicht (204-1) des neuronalen Netzwerks einen ersten Verzögerungsfaktor aufweist, der sich von einem zweiten Verzögerungsfaktor der zweiten verborgenen Schicht des neuronalen Netzwerks unterscheidet;

Empfangen, durch eine Ausgabeschicht (204-(N-1)) des neuronalen Netzwerks, die auf die erste verborgene Schicht (204-1) des neuronalen Netzwerks von einer vorstehenden Schicht des neuronalen Netzwerks der Ausgabeschicht (204-(N-1)) des neuronalen Netzwerks folgt, einer zweiten Eingabeaktivierung; und

Bereitstellen, durch die Ausgabeschicht (204-(N-1)) des neuronalen Netzwerks an die vorstehende Schicht des neuronalen Netzwerks, eines ersten Ausgabegradients basierend auf der zweiten Eingabeaktivierung und dem ersten Trainingssatz (220), wobei der erste Ausgabegradients (208-N...208-1) unter Verwendung des einen oder der mehreren Rückwärtspfadverarbeitungselemente (310-0...310-(P-1)) berechnet wird.

13. Verfahren nach Anspruch 12, ferner umfassend:

Bereitstellen, durch die erste verborgene Schicht (204-1) des neuronalen Netzwerks an die vorstehende Schicht des neuronalen Netzwerks, eines zweiten Ausgabegradients, der mit dem zweiten Stapel der Folge von Stapeln verknüpft ist, basierend auf der ersten Eingabeaktivierung und dem ersten Eingabegradients.

14. Verfahren nach Anspruch 13, ferner umfassend:

Erzeugen, durch die erste verborgene Schicht (204-1) des neuronalen Netzwerks, einer ersten Wertung, die mit dem zweiten Stapel der Folge von Stapeln verknüpft ist, basierend auf einer dritten Eingabeaktivierung, die mit dem zweiten Stapel und dem ersten Eingabegradients verknüpft ist; und

Erzeugen, durch die erste verborgene Schicht (204-1) des neuronalen Netzwerks, der ersten Ausgabeaktivierung, die mit dem ersten Stapel der Folge von Stapeln verknüpft ist, basierend auf der ersten Eingabeaktivierung, dem ersten Eingabegradients und der ersten Wertung.

15. Verfahren nach Anspruch 14, ferner umfassend:

Durchführen von mindestens einem Vorgang, der ausgewählt ist aus:

Erzeugen, durch die erste verborgene Schicht (204-1) des neuronalen Netzwerks, des zweiten Ausgabegradients basierend auf der ersten Eingabeaktivierung, dem ersten Eingabegradi-

enten und der ersten Wertung; und
 Anpassen, durch die erste verborgene Schicht
 (204-1) des neuronalen Netzwerks, einer Lern-
 rate basierend auf dem Verzögerungsfaktor zwi- 5
 schen dem ersten Stapel und dem zweiten Sta-
 pel und Erzeugen, durch die erste verborgene
 Schicht (204-1) des neuronalen Netzwerks, der
 ersten Wertung, die mit dem zweiten Stapel der
 Folge von Stapeln verknüpft ist, durch Anwen- 10
 den der Lernrate an dem ersten Eingabegradi-
 enten, der mit dem zweiten Stapel verknüpft ist.

Revendications

1. Système de réseau neuronal (200) comprenant :
 un ou plusieurs processeurs matériels configurés
 pour exécuter des instructions pour amener le sys-
 tème à mettre en oeuvre un modèle de réseau neu-
 ronal, dans lequel la mise en oeuvre du modèle de 20
 réseau neuronal comprend :

des éléments de traitement de chemin vers
 l'avant (306-0...306(P-1)) et des éléments de
 traitement de chemin vers l'arrière 25
 (310-0...310-(P-1)),
 dans lequel chaque neurone du modèle de ré-
 seau neuronal possède un élément de traite-
 ment de chemin vers l'avant (306-0...306(P-1))
 correspondant et un élément de traitement de 30
 chemin vers l'arrière (310-0...310-(P-1)) corres-
 pondant,
 dans lequel le système (200) est en outre con-
 figuré pour :
 mettre en oeuvre une couche de réseau neuro- 35
 nal d'entrée (204-0) configurée pour :

recevoir un premier ensemble d'entraîne-
 ment (220) comportant une séquence de
 lots ; et 40

fournir, à une couche de réseau neuronal
 suivante, une pluralité d'activations de sor-
 tie (206-1 - 206-(N-1)) associées à la sé-
 quence de lots dans un chemin vers l'avant,
 respectivement, dans lequel le chemin vers 45
 l'avant dans la couche de réseau neuronal
 suivante comporte le ou les éléments de
 traitement de chemin vers l'avant
 (306-0...306(P-1)), dans lequel la pluralité
 d'activations de sortie (206-1...206(N-1)) 50
 est calculée à l'aide du ou des éléments de
 traitement de chemin vers l'avant
 (306-0...306(P-1)) ;

mettre en oeuvre de première et seconde
 couches de réseau neuronal cachées 55
 (204-1 ; 204-2) suivant la couche de réseau
 neuronal d'entrée (204-0), chaque couche
 de réseau neuronal cachée étant configu-

rée pour :

recevoir, à partir d'une couche de ré-
 seau neuronal précédente, une pre-
 mière activation d'entrée associée à un
 premier lot de la séquence de lots ;
 réaliser, à l'aide des éléments de trai-
 tement de chemin vers l'avant (306-0,
 306-(P-1)), une fonction de génération
 d'activations pour générer des activa-
 tions (206-i) sur la base d'activations
 (204-i) reçues à partir des éléments de
 traitement de chemin vers l'avant
 (306-0, 306-(P-1)) correspondant aux
 neurones de la couche précédente
 (204-(i-1)), et envoyer les activations
 générées (206-i) aux éléments de trai-
 tement de chemin vers l'avant (306-0,
 306-(P-1)) correspondant aux neuro-
 nes de la couche suivante (204-(i+1)),
 recevoir, lorsque la première activation
 d'entrée est reçue, un premier gradient
 d'entrée (208-(i+1)) calculé dans le ou
 les éléments de traitement de chemin
 vers l'arrière (310-0...310-(P-1)) d'une
 couche de réseau neuronal subsé-
 quente et associé à un deuxième lot de
 la séquence de lots précédant le pre-
 mier lot par un facteur de retard (d_i) as-
 socié au premier lot, les calculs dans le
 ou les éléments de traitement de che-
 min vers l'arrière (310-0...310-(P-1))
 dans chaque couche de réseau neuro-
 nal cachée étant mis en oeuvre paral-
 lèlement aux calculs dans le ou les élé-
 ments de traitement de chemin vers
 l'avant ;

dans lequel le facteur de retard (d_i) cor-
 respond à une latence de deux lots ou
 plus autres que les premier et deuxiè-
 me lots ; et

réaliser, à l'aide des éléments de trai-
 tement de chemin vers l'arrière (310-0,
 310-(P-1)), une fonction de génération
 de gradients pour générer des gra-
 dients (208-i) sur la base de la première
 activation d'entrée reçue (206-(i-1)) et
 du premier gradient d'entrée reçu
 (208-(i+1)) ;

fournir, à la couche de réseau neuronal
 précédente, les gradients générés
 (208-i) ;

fournir, à une couche de réseau neuro-
 nal suivante de la couche de réseau
 neuronal cachée, une première activa-
 tion de sortie associée au premier lot,
 dans lequel la première activation de
 sortie est déterminée sur la base de la

- première activation d'entrée et du premier gradient d'entrée ;
 dans lequel la première couche de réseau neuronal cachée (204-1) a un premier facteur de retard qui est différent d'un second facteur de retard de la seconde couche cachée ;
 une couche de sortie de réseau neuronal (204-(N-1)) suivant les première et seconde couches de réseau neuronal cachées, dans lequel la couche de sortie (204-(N-1)) est configurée pour :
- recevoir, à partir d'une couche de réseau neuronal précédente, une deuxième activation d'entrée ; et
 fournir, à la couche de réseau neuronal précédente, un premier gradient de sortie (208-N - 208-1) sur la base de la deuxième activation d'entrée et du premier ensemble d'entraînement (220), dans lequel le premier gradient de sortie (208-N...208-1) est calculé à l'aide du ou des éléments de traitement de chemin vers l'arrière (310-0...310-(P-1)).
2. Système de réseau neuronal (200) selon la revendication 1, dans lequel la première couche cachée (204-1) est configurée pour :
 fournir, à la couche de réseau neuronal précédente, un second gradient de sortie associé au deuxième lot de la séquence de lots sur la base de la première activation d'entrée et du premier gradient d'entrée, dans lequel le second gradient de sortie (208-N...208-1) est calculé à l'aide du ou des éléments de traitement de chemin vers l'arrière (310-0...310-(P-1)).
3. Système de réseau neuronal (200) selon la revendication 2, dans lequel la première couche de réseau neuronal cachée (204-1) est configurée pour :
 générer un premier poids associé au deuxième lot de la séquence de lots sur la base d'une troisième activation d'entrée associée au deuxième lot de la séquence de lots et au premier gradient d'entrée ; et
 générer la première activation de sortie associée au premier lot de la séquence de lots sur la base de la première activation d'entrée, du premier gradient d'entrée, et du premier poids.
4. Système de réseau neuronal (200) selon la revendication 3, dans lequel la première couche de réseau neuronal cachée (204-1) est configurée pour :
 générer le second gradient de sortie sur la base de
- la première activation d'entrée, du premier gradient d'entrée, et du premier poids.
5. Système de réseau neuronal (200) selon la revendication 3, dans lequel la première couche de réseau neuronal cachée (204-1) est configurée pour :
 ajuster un taux d'apprentissage sur la base du facteur de retard entre le premier lot de la séquence de lots et le deuxième lot de la séquence de lots ; et
 générer le premier poids associé au deuxième lot de la séquence de lots en appliquant le taux d'apprentissage au premier gradient d'entrée associé au deuxième lot de la séquence de lots.
6. Système de réseau neuronal (200) selon la revendication 1, comprenant en outre :
 une unité de prétraitement (202) configurée pour :
 recevoir un ensemble d'entraînement original (220) ayant une première taille ; et
 générer le premier ensemble d'entraînement sur la base de l'ensemble d'entraînement original (220), dans lequel le premier ensemble d'entraînement a une seconde taille supérieure à la première taille.
7. Système de réseau neuronal (200) selon la revendication 1, dans lequel la première couche de réseau neuronal cachée (204-1) est couplée à un générateur de gradient synthétique (902) qui est configuré pour :
 générer une première estimation de gradient d'entrée à l'aide d'un second gradient d'entrée associé à un troisième lot précédant le deuxième lot de la séquence de lots ; et
 fournir à la première couche cachée (204-1) la première estimation de gradient d'entrée en tant que premier gradient d'entrée.
8. Système de réseau neuronal (200) selon la revendication 7, dans lequel le générateur de gradient synthétique (902) est configuré pour :
 recevoir un second gradient d'entrée provenant de la couche de réseau neuronal suivante ;
 générer une erreur d'estimation de gradient sur la base de la première estimation de gradient d'entrée et du second gradient d'entrée ; et
 générer une seconde estimation de gradient d'entrée sur la base de l'erreur d'estimation de gradient à l'aide d'un algorithme d'apprentissage automatique.
9. Système de réseau neuronal (200) selon la revendication 1, dans lequel la première activation d'en-

trée, la première activation de sortie, la deuxième activation d'entrée, et le premier gradient de sortie comportent des valeurs binaires.

10. Système de réseau neuronal (200) selon la revendication 1, comprenant en outre :
une représentation de données logarithmiques configurée pour quantifier la première activation d'entrée, la première activation de sortie, la deuxième activation d'entrée, et le premier gradient de sortie dans un domaine logarithmique. 5 10
11. Système de réseau neuronal (200) selon la revendication 1, comprenant en outre : 15
une pluralité d'accélérateurs,
dans lequel chaque accélérateur est configuré pour mettre en oeuvre une couche de la couche d'entrée, la ou les couches cachées, et la couche de sortie (204-(N-1)), et 20
dans lequel chaque accélérateur est configuré pour avoir une capacité de calcul et une capacité de mémoire basées sur la couche respective.
12. Procédé mis en oeuvre par ordinateur destiné à mettre en oeuvre un modèle de réseau neuronal, dans lequel la mise en oeuvre du modèle de réseau neuronal comprend des éléments de traitement de chemin vers l'avant (306-0...306(P-1)) et des éléments de traitement de chemin vers l'arrière (310-0...310-(P-1)), dans lequel chaque neurone du modèle de réseau neuronal possède un élément de traitement de chemin vers l'avant (306-0...306(P-1)) correspondant et un élément de traitement de chemin vers l'arrière (310-0...310-(P-1)) correspondant, dans lequel le procédé mis en oeuvre par ordinateur comprend : 25 30 35
la réception, par une couche de réseau neuronal d'entrée (204-0) d'un réseau neuronal, d'un premier ensemble d'entraînement (220) comportant une séquence de lots ; 40
la fourniture, par la couche de réseau neuronal d'entrée (204-0) à une couche de réseau neuronal suivante, d'une pluralité d'activations de sortie associées à la séquence de lots dans un chemin vers l'avant, respectivement, dans lequel le chemin vers l'avant dans la couche de réseau neuronal suivante comporte le ou les éléments de traitement de chemin vers l'avant (306-0...306(P-1)), dans lequel la pluralité d'activations de sortie (206-1...206(N-1)) est calculée à l'aide du ou des éléments de traitement de chemin vers l'avant (306-0...306(P-1)) ; 45 50
la réception, par une première couche de réseau neuronal cachée (204-1) du réseau neuronal à partir d'une couche de réseau neuronal précédente, d'une première activation d'entrée asso- 55

ciée à un premier lot de la séquence de lots ;
la réalisation, à l'aide des éléments de traitement de chemin vers l'avant (306-0, 306-(P-1)), d'une fonction de génération d'activations pour générer des activations (206-i) sur la base d'activations (204-i) reçues à partir des éléments de traitement de chemin vers l'avant (306-0, 306-(P-1)) correspondant aux neurones de la couche précédente (204-(i-1)), et envoyer les activations générées (206-i) aux éléments de traitement de chemin vers l'avant (306-0, 306-(P-1)) correspondant aux neurones de la couche suivante (204-(i+1)),
la réception, par la première couche de réseau neuronal cachée (204-1) lorsque la première activation d'entrée est reçue, d'un premier gradient d'entrée (208-(i+1)) calculé dans le ou les éléments de traitement de chemin vers l'arrière (310-0...310-(P-1)) d'une couche de réseau neuronal subséquente et associé à un deuxième lot de la séquence de lots précédant le premier lot par un facteur de retard (d_i) associé au premier lot, dans lequel les calculs dans le ou les éléments de traitement de chemin vers l'arrière (310-0...310-(P-1)) dans chaque couche cachée sont mis en oeuvre parallèlement aux calculs dans le ou les éléments de traitement de chemin vers l'avant et le facteur de retard (d_i) correspond à une latence de deux lots ou plus autres que les premier et deuxième lots de la séquence de lots ;
la réalisation, à l'aide des éléments de traitement de chemin vers l'arrière (310-0, 310-(P-1)), d'une fonction de génération de gradients pour générer des gradients (208-i) sur la base de la première activation d'entrée reçue (206-(i-1)) et du premier gradient d'entrée reçu (208-(i+1)) ;
la fourniture, à la couche de réseau neuronal précédente, des gradients générés (208-i) ;
la fourniture, par la première couche de réseau neuronal cachée (203-1) à une seconde couche de réseau neuronal cachée (204-2) suivant la première couche cachée (204-1), d'une première activation de sortie associée au premier lot, dans lequel la première activation de sortie est basée sur la première activation d'entrée et le premier gradient d'entrée ;
dans lequel la première couche de réseau neuronal cachée (204-1) a un premier facteur de retard qui est différent d'un second facteur de retard de la seconde couche de réseau neuronal cachée ;
la réception, par une couche de réseau neuronal de sortie (204-(N-1)) suivant la première couche de réseau neuronal cachée (204-1), à partir d'une couche de réseau neuronal précédente de la couche de réseau neuronal de sortie

(204-(N-1)), d'une deuxième activation d'entrée ; et
 la fourniture, par la couche de réseau neuronal de sortie (204-(N-1)) à la couche de réseau neuronal précédente, d'un premier gradient de sortie sur la base de la deuxième activation d'entrée et du premier ensemble d'entraînement (220), dans lequel le premier gradient de sortie (208-N... 208-1) est calculé à l'aide du ou des éléments de traitement de chemin vers l'arrière (310-0...310-(P-1)).

13. Procédé selon la revendication 12, comprenant en outre :
 la fourniture, par la première couche de réseau neuronal cachée (204-1) à la couche de réseau neuronal précédente, d'un second gradient de sortie associé au deuxième lot de la séquence de lots sur la base de la première activation d'entrée et du premier gradient d'entrée.
14. Procédé selon la revendication 13, comprenant en outre :
 la génération, par la première couche de réseau neuronal cachée (204-1), d'un premier poids associé au deuxième lot de la séquence de lots sur la base d'une troisième activation d'entrée associée au deuxième lot et au premier gradient d'entrée ; et
 la génération, par la première couche de réseau neuronal cachée (204-1), de la première activation de sortie associée au premier lot de la séquence de lots sur la base de la première activation d'entrée, du premier gradient d'entrée, et du premier poids.
15. Procédé selon la revendication 14, comprenant en outre :
 la réalisation d'au moins une opération choisie parmi :
 la génération, par la première couche de réseau neuronal cachée (204-1), du second gradient de sortie sur la base de la première activation d'entrée, du premier gradient d'entrée, et du premier poids ; et
 l'ajustement, par la première couche de réseau neuronal cachée (204-1), d'un taux d'apprentissage sur la base du facteur de retard entre le premier lot et le deuxième lot et la génération, par la première couche de réseau neuronal cachée (204-1), du premier poids associé au deuxième lot de la séquence de lots en appliquant le taux d'apprentissage au premier gradient d'entrée associé au deuxième lot.

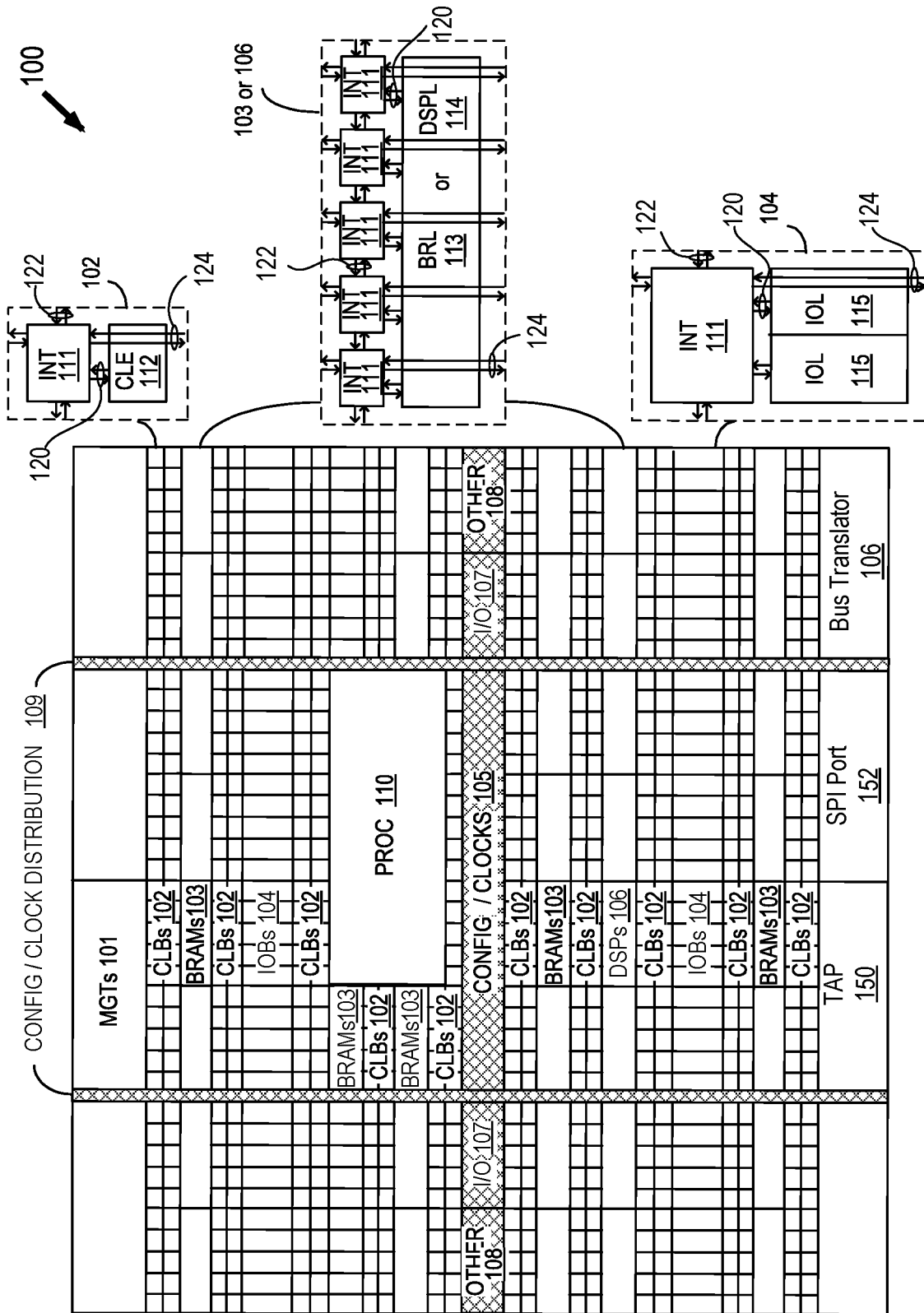


FIG. 1

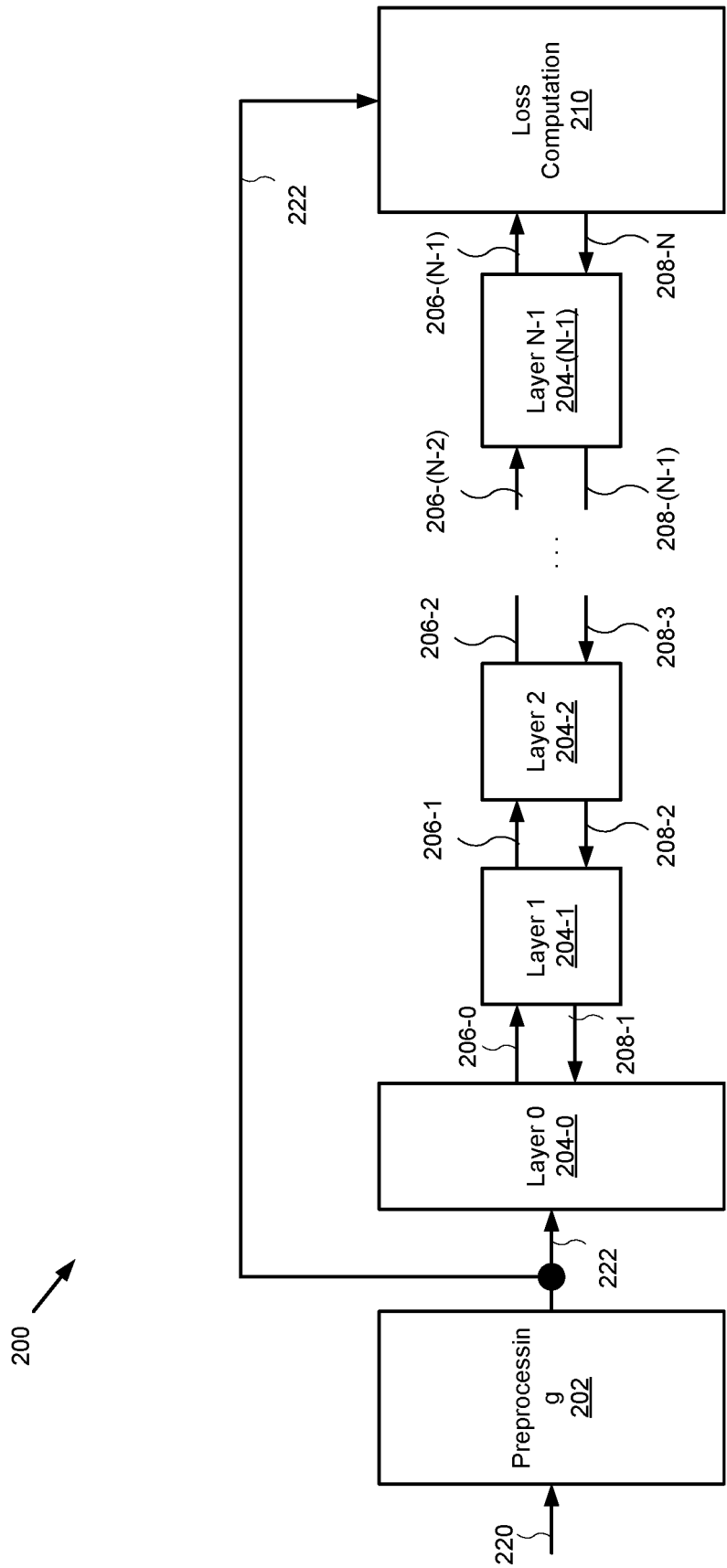


FIG. 2

204-i
↙

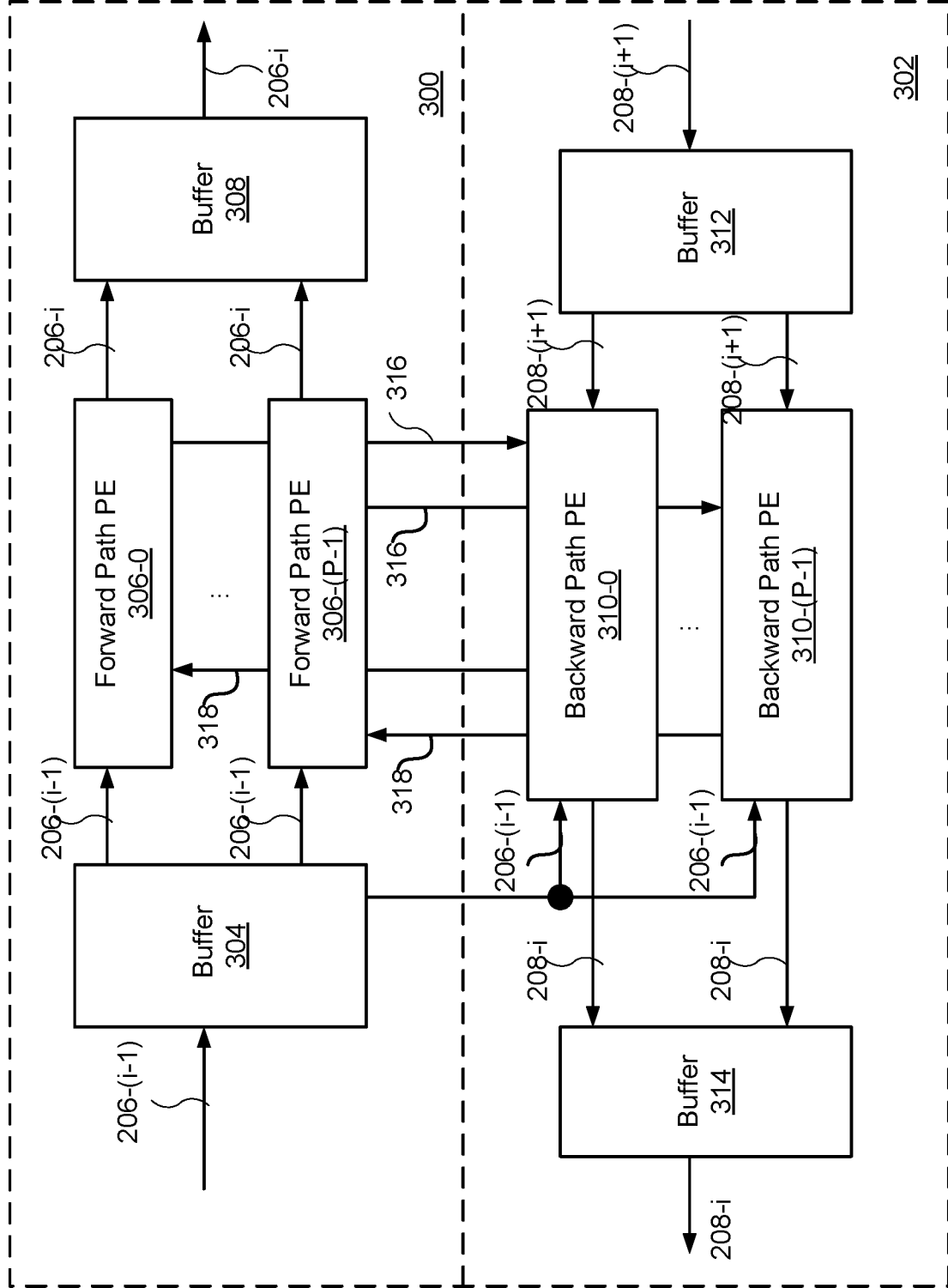


FIG. 3

306-p

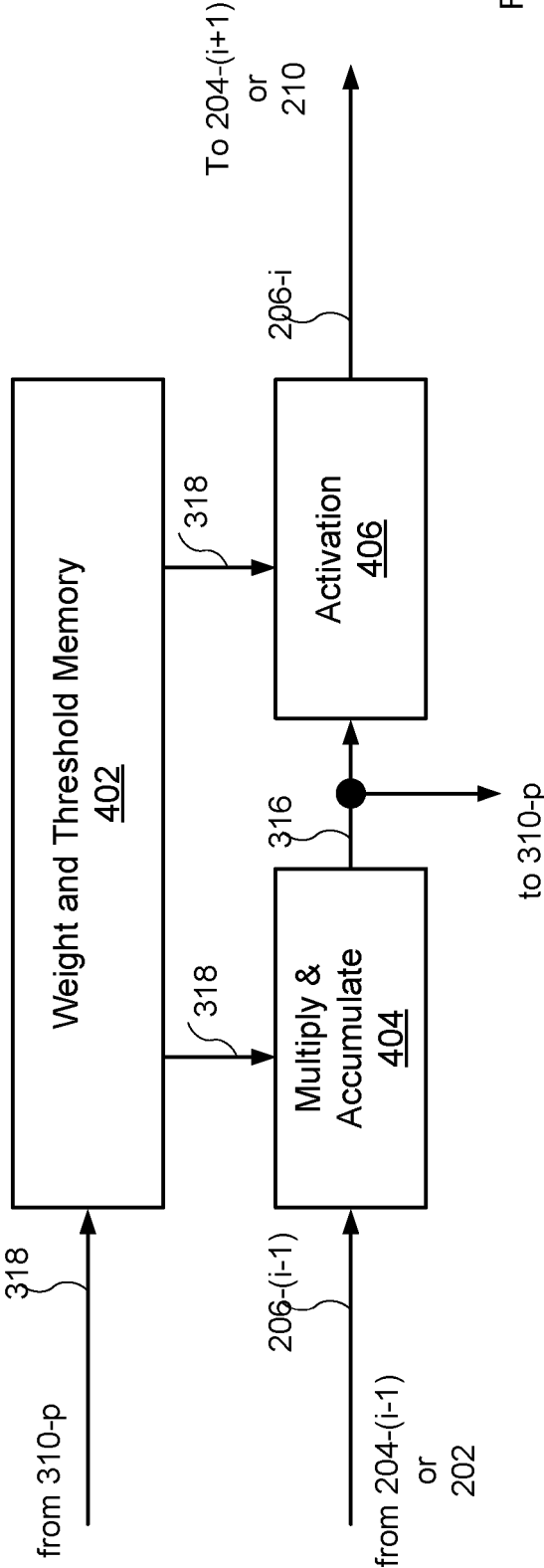


FIG. 4

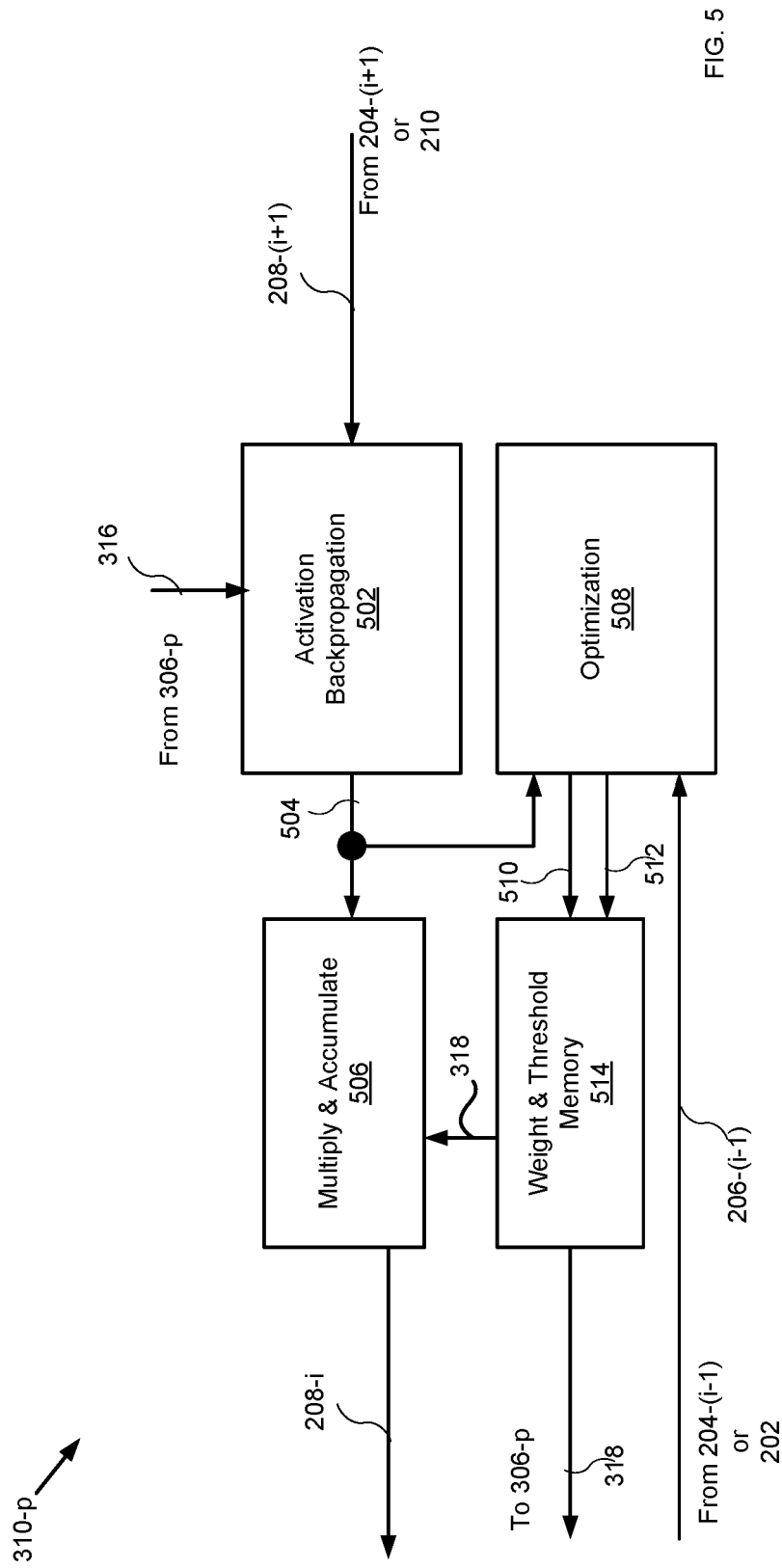


FIG. 5

600 ↗

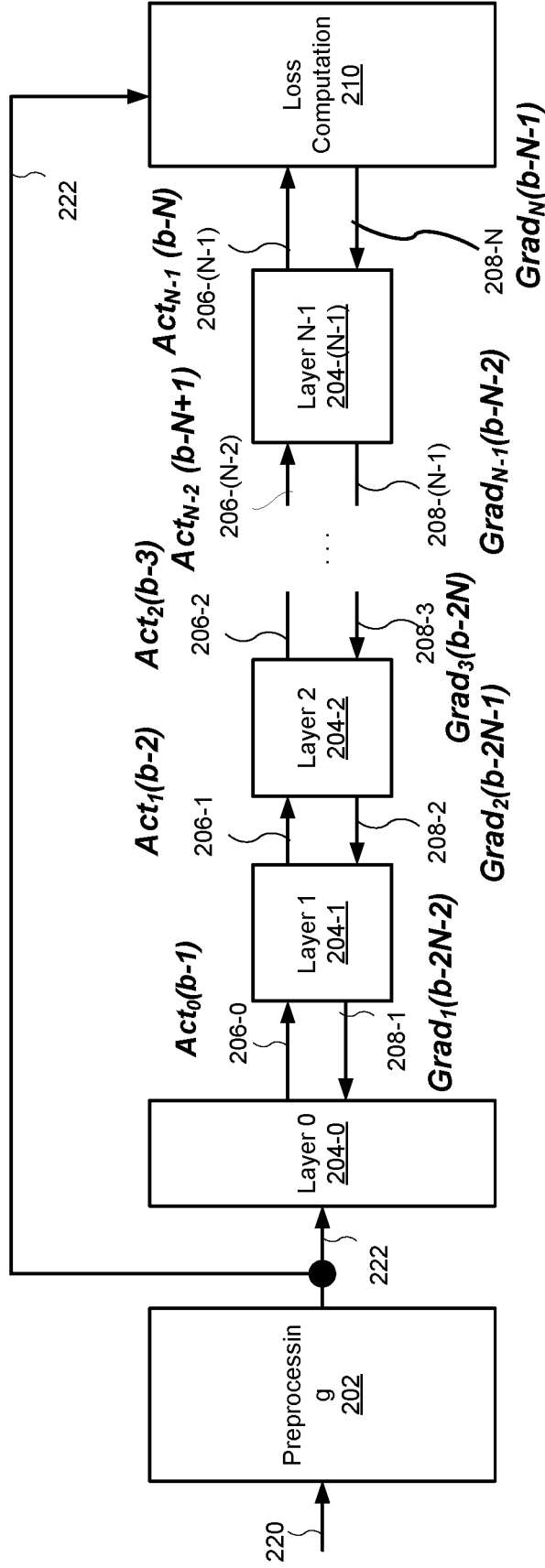


FIG. 6

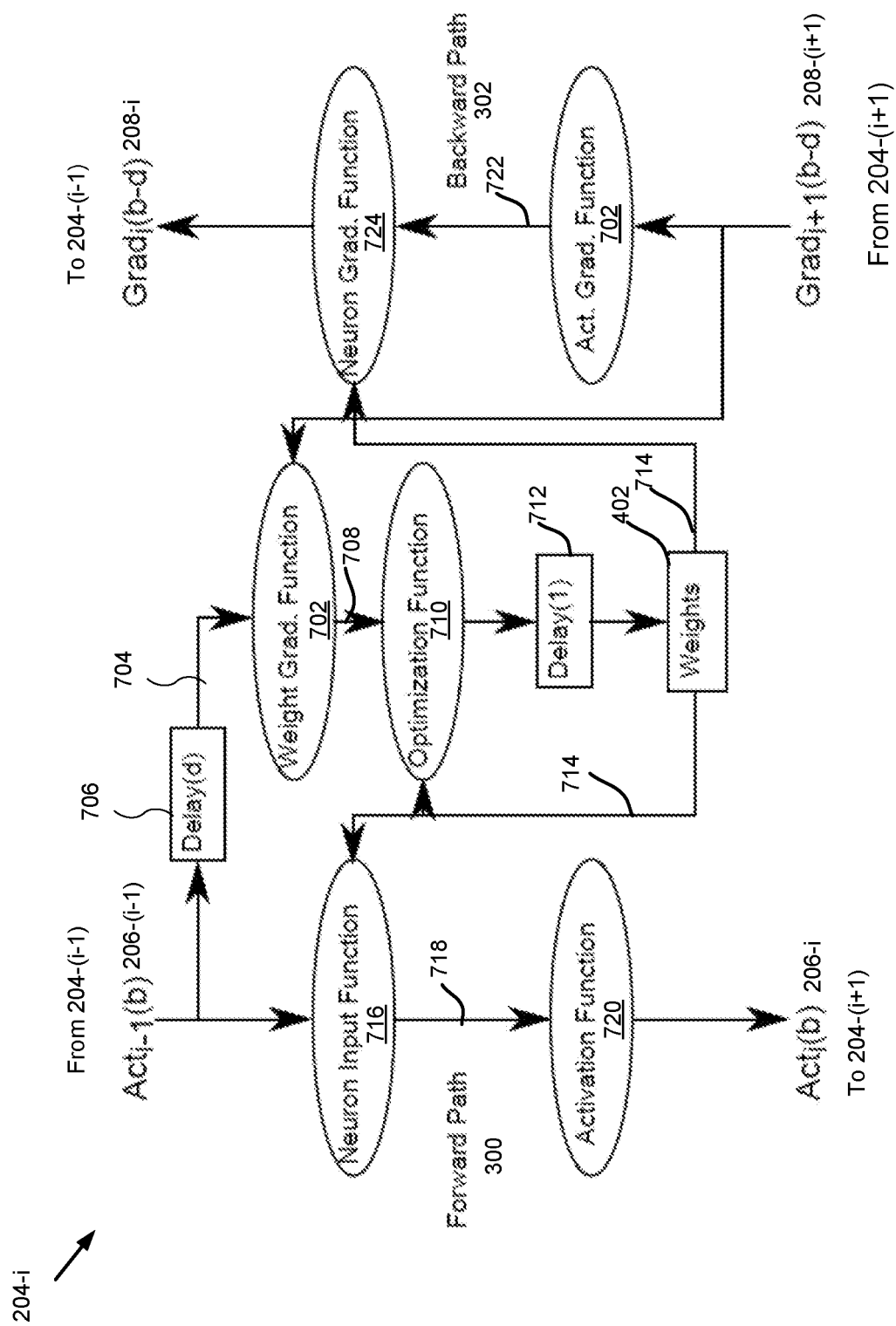


FIG. 7

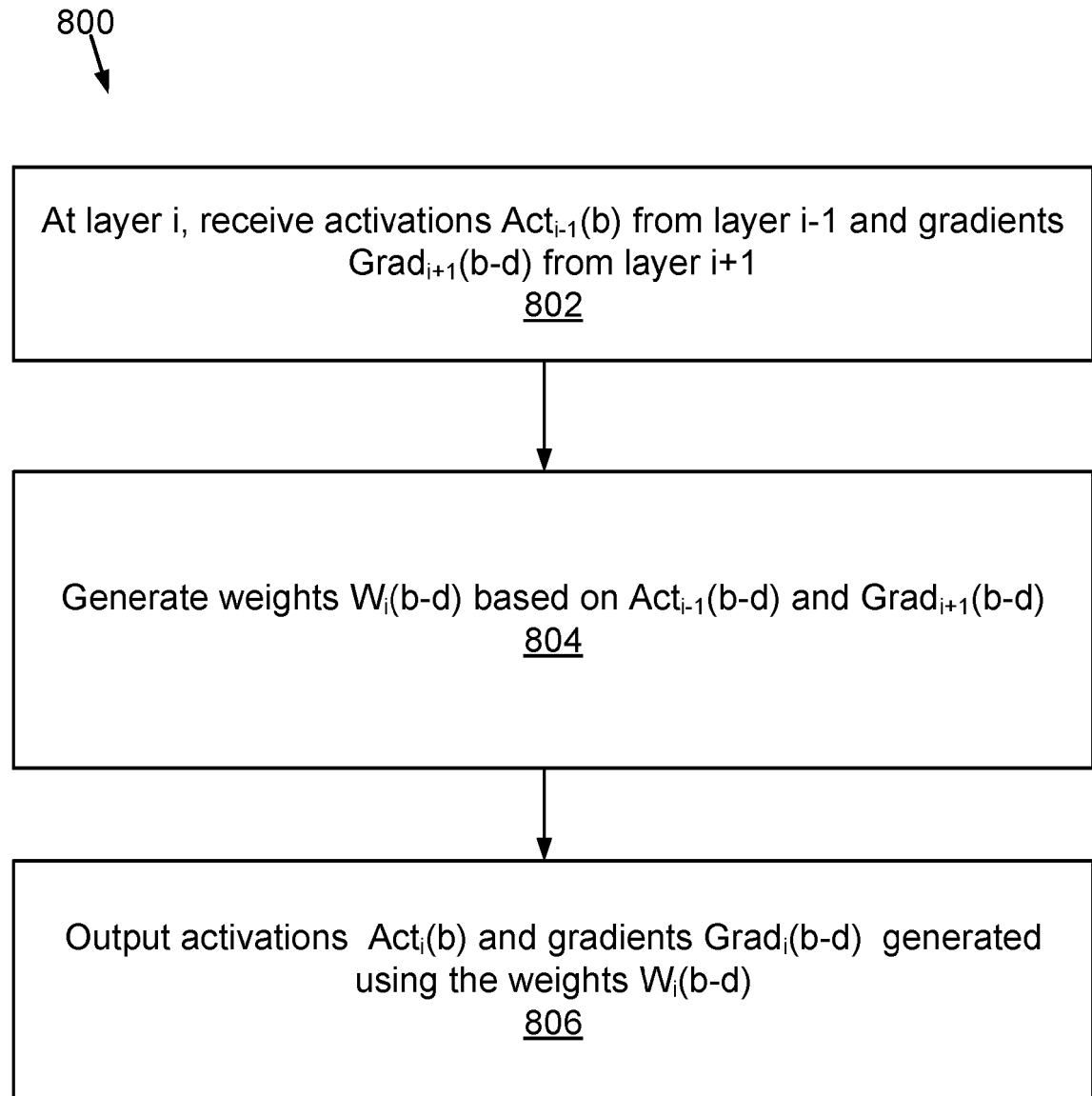


FIG. 8

310-p ↗

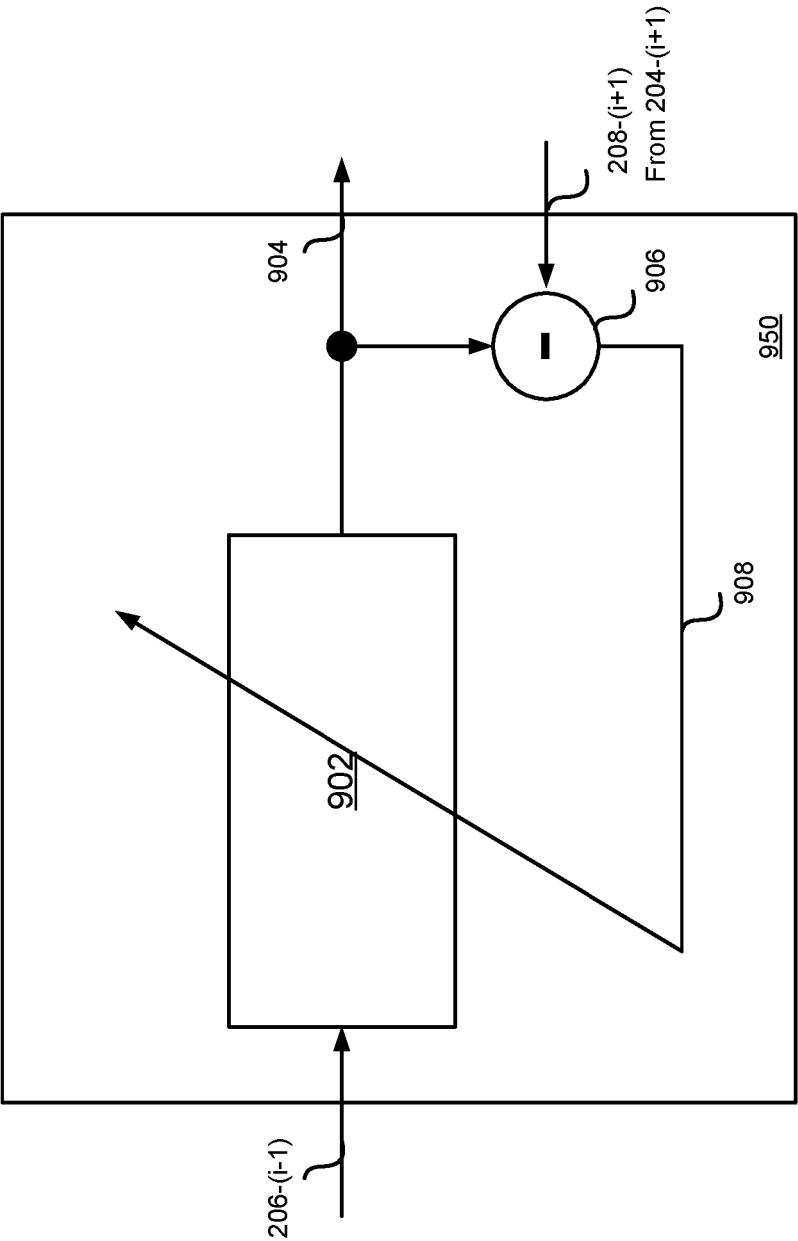


FIG. 9A

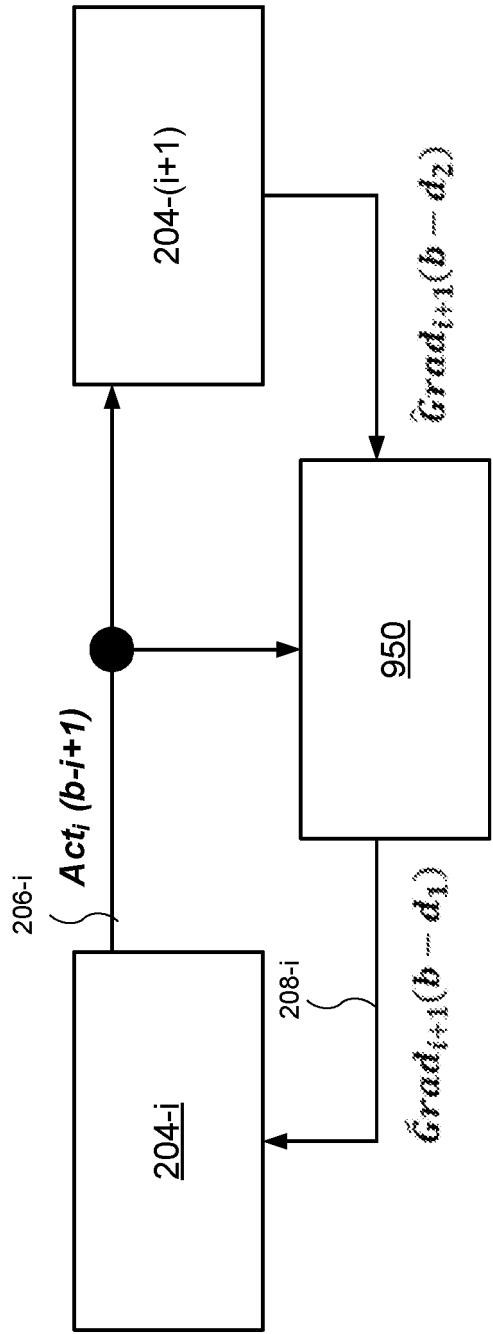


FIG. 9B

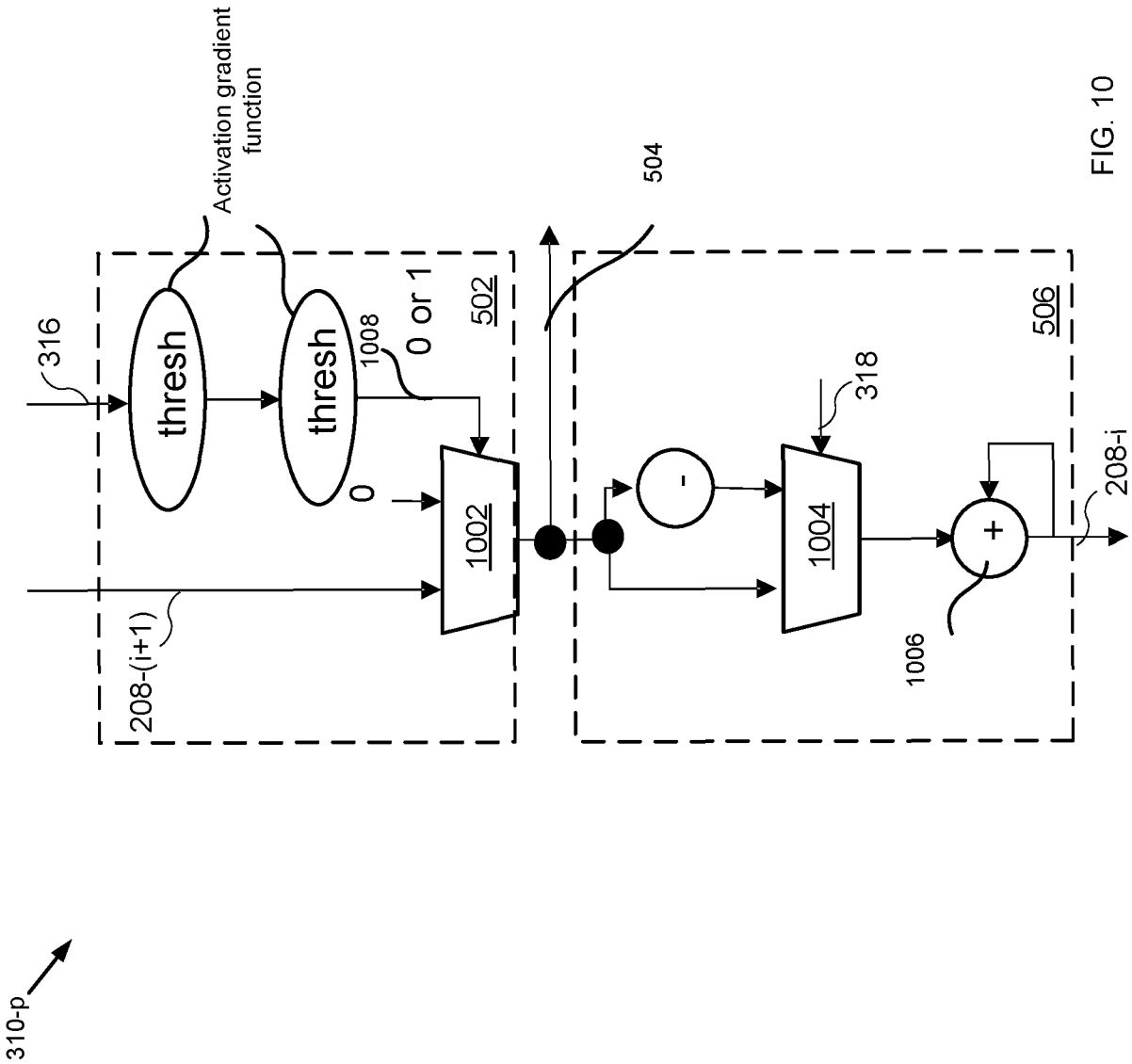


FIG. 10

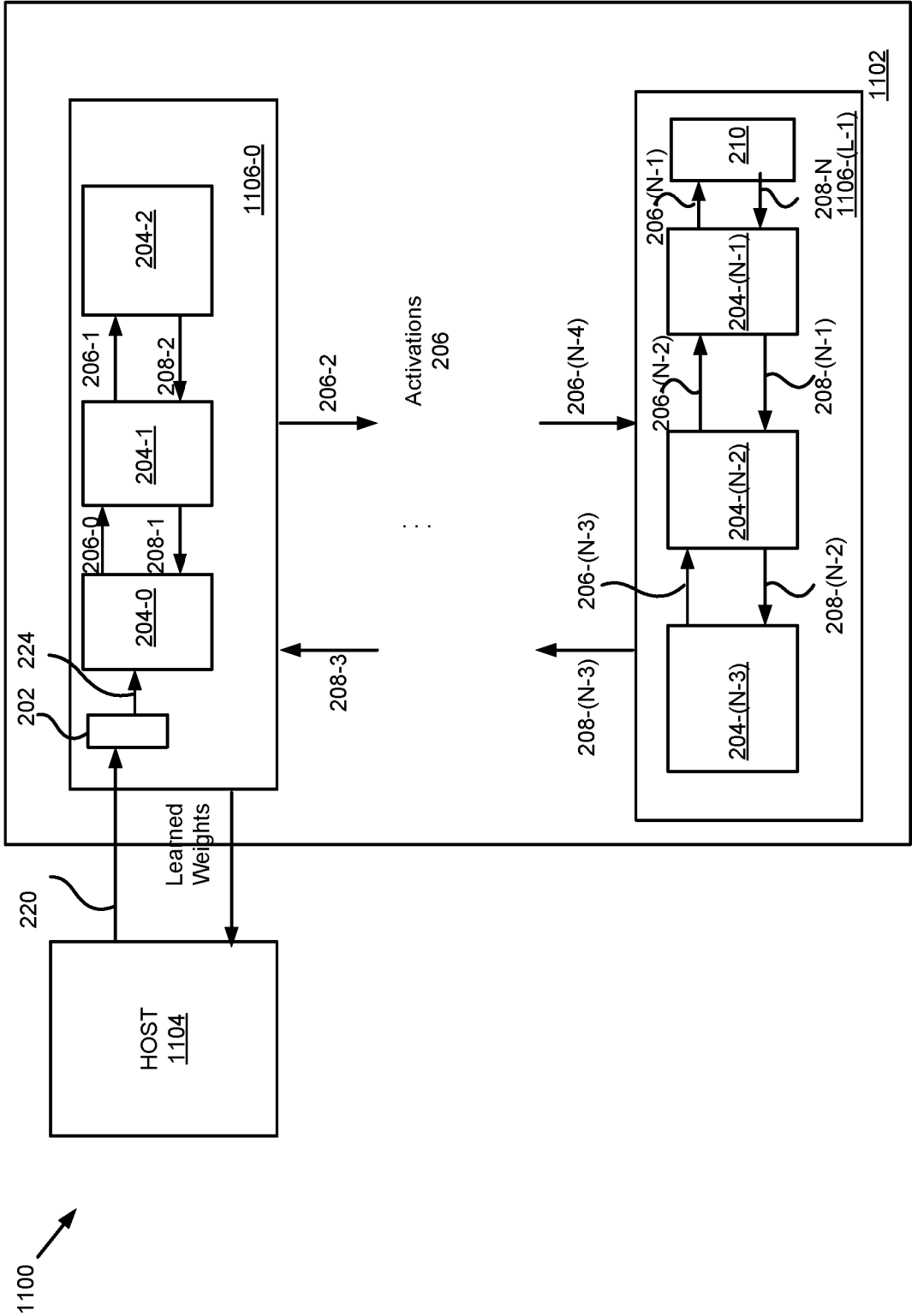


FIG. 11

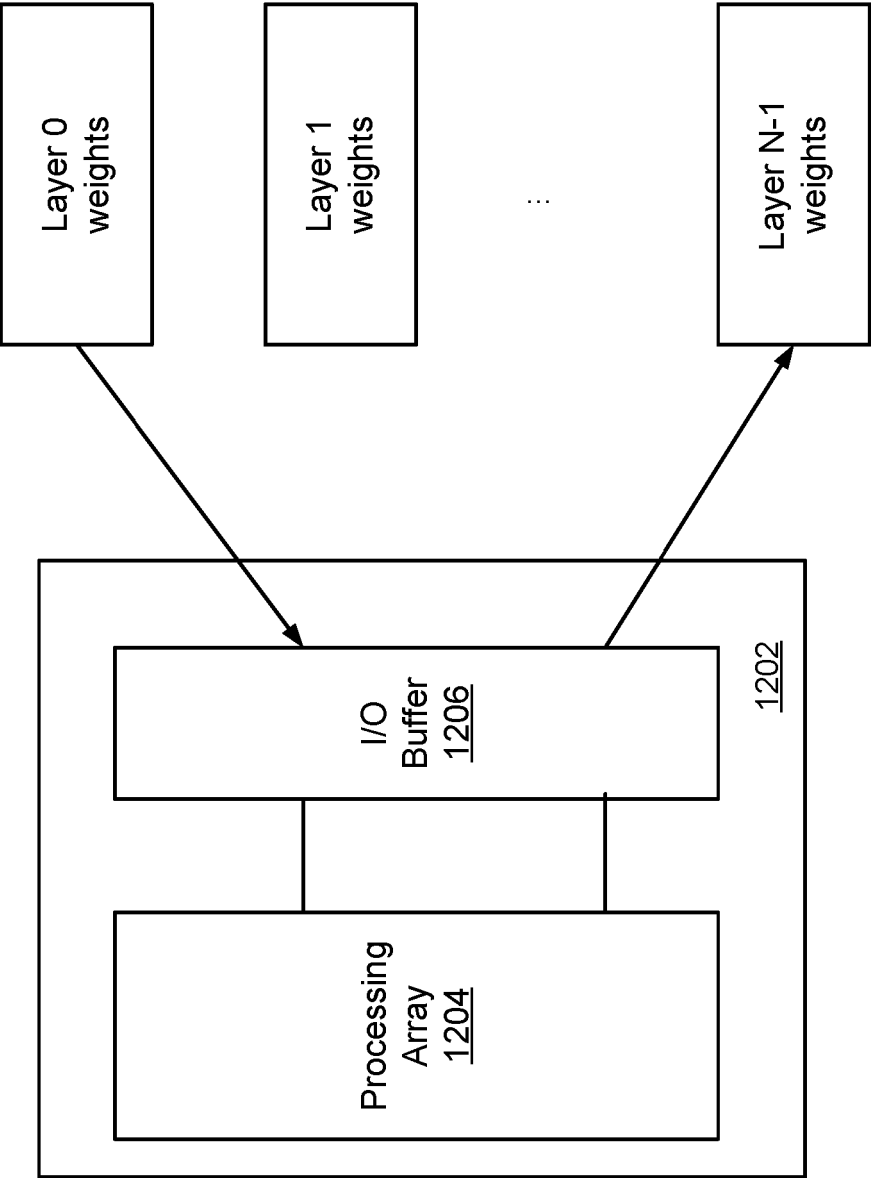


FIG. 12

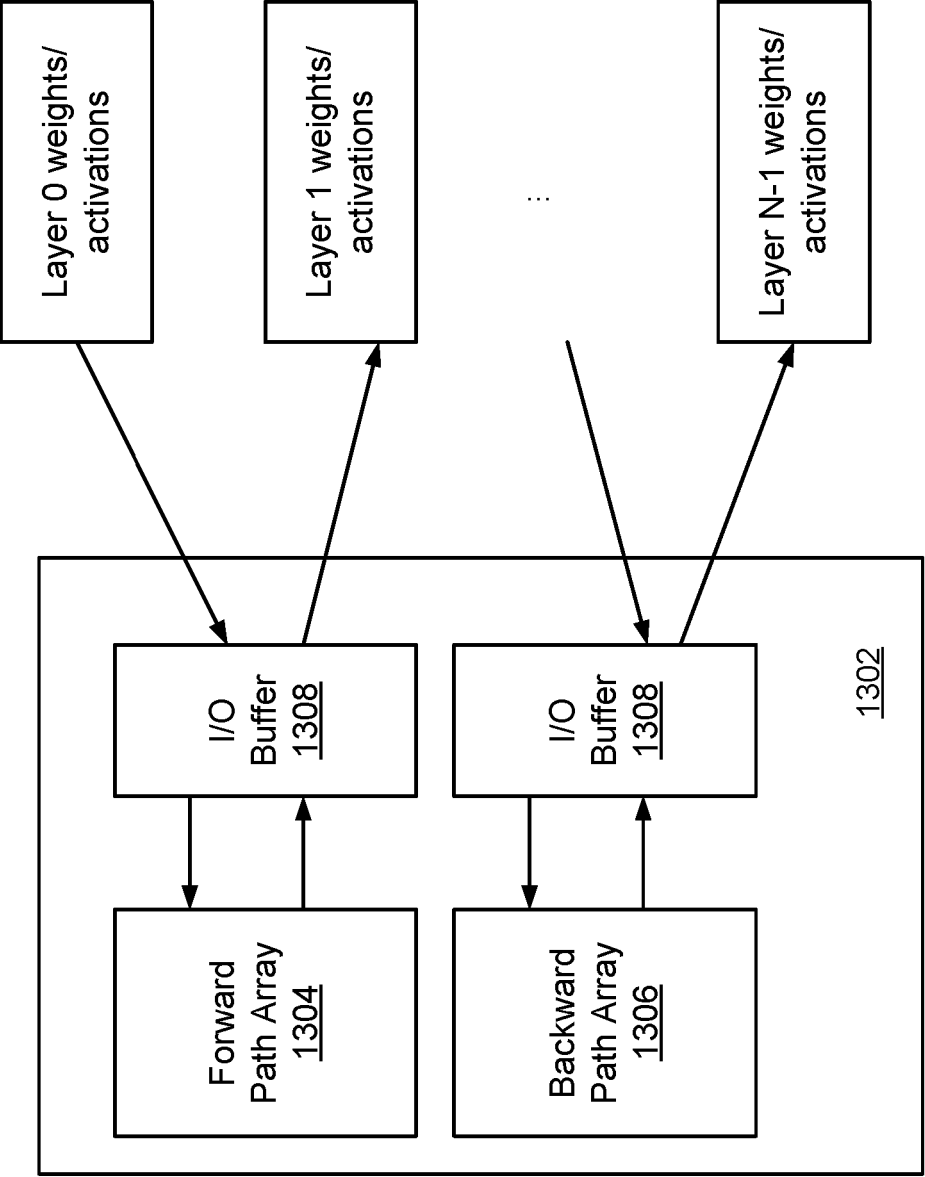


FIG. 13

REFERENCES CITED IN THE DESCRIPTION

This list of references cited by the applicant is for the reader's convenience only. It does not form part of the European patent document. Even though great care has been taken in compiling the references, errors or omissions cannot be excluded and the EPO disclaims all liability in this regard.

Non-patent literature cited in the description

- Decoupled Neural Interfaces using Synthetic Gradients. **MAX JADERBERG et al.** ARXIV.ORG. CORNELL UNIVERSITY LIBRARY, 18 August 2016
[0003]